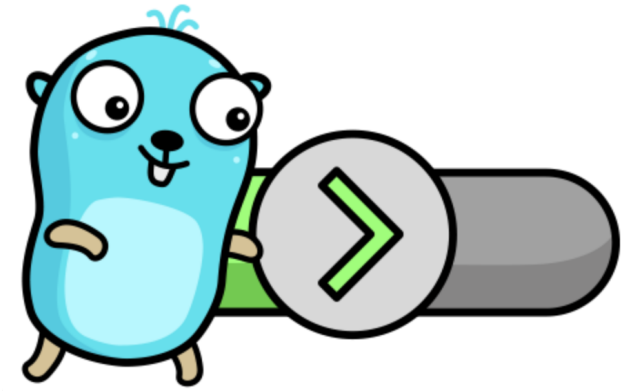
Must-функции

В этом уроке мы поговорим о негласных правилах нейминга паникующих функций, а также лучших практиках их реализации и использования.



Must!

В Go существует паттерн, согласно которому функции и методы, которые при ошибке будут паниковать (или завершать процесс с помощью `log.Fatal`/`os.Exit`), должны начинаться с префикса `Must` (`must`).

Никаких синтаксических гарантий или проверок от компилятора префикс `Must` не несёт – это просто договорённость между разработчиками на уровне написания и чтения кода:

```
// https://github.com/golang/go master
$ grep --exclude "*test*" -E "func (\(\w{1,3} \*?\w+\)
)?(m|M)ust.*\(" -R .

./cmd/go/internal/modload/build.go
    func mustFindModule(ld *loader, target, path string)
module.Version {

./cmd/internal/bio/buf.go
    func (r *Reader) MustSeek(offset int64, whence int) int64 {

./cmd/internal/bio/buf.go
    func (w *Writer) MustSeek(offset int64, whence int) int64 {
```

```
./cmd/internal/bio/must.go
    func MustClose(c io.Closer) {

./cmd/internal/bio/must.go
    func MustWriter(w io.Writer) io.Writer {

./cmd/internal/sys/supported.go
    func MustLinkExternal(goos, goarch string) bool {

./cmd/link/internal/ld/config.go
    func mustLinkExternal(ctxt *Link) (res bool, reason string) {

./cmd/link/internal/ld/target.go
    func (t *Target) mustSetHeadType() {

./cmd/link/internal/ld/dwarf.go
    func (d *dwctxt) mustFind(name string) loader.Sym {

./cmd/vendor/golang.org/x/sys/windows/dll_windows.go
    func MustLoadDLL(name string) *DLL {

./cmd/vendor/golang.org/x/sys/windows/dll_windows.go
    func (d *DLL) MustFindProc(name string) *Proc {

./cmd/vendor/golang.org/x/sys/windows/dll_windows.go
    func (d *DLL) MustFindProcByOrdinal(ordinal uintptr) *Proc {

./cmd/vendor/golang.org/x/sys/windows/dll_windows.go
    func (d *LazyDLL) mustLoad() {

./cmd/vendor/golang.org/x/sys/windows/dll_windows.go
    func (p *LazyProc) mustFind() {

./cmd/vendor/golang.org/x/mod/modfile/rule.go
    func MustQuote(s string) bool {

./crypto/ed25519/internal/edwards25519/field/_asm/fe_amd64_asm.go
    func mustAddr(c Component) Op {

./net/textproto/reader.go
    func mustHaveFieldNameColon(line []byte) error {

./net/addrselect.go
    func mustCIDR(s string) *IPNet {
```

```
./regexp/regexp.go
    func MustCompile(str string) *Regexp {

./regexp/regexp.go
    func MustCompilePOSIX(str string) *Regexp {

./archive/tar/format.go
    func (f *Format) mustNotBe(f2 Format) { *f &^= f2 }

./archive/tar/reader.go
    func mustReadFull(r io.Reader, b []byte) (int, error) {

./fmt/scan.go
    func (s *ss) mustReadRune() (r rune) {

./reflect/value.go
    func (f flag) mustBe(expected Kind) {

./reflect/value.go
    func (f flag) mustBeExported() {

./reflect/value.go
    func (f flag) mustBeExportedSlow() {

./reflect/value.go
    func (f flag) mustBeAssignable() {

./reflect/value.go
    func (f flag) mustBeAssignableSlow() {

./internal/reflectlite/value.go
    func (f flag) mustBeExported() {

./internal/reflectlite/value.go
    func (f flag) mustBeAssignable() {

./encoding/xml/xml.go
    func (d *Decoder) mustgetc() (b byte, ok bool) {

./encoding/gob/type.go
    func mustGetTypeInfo(rt reflect.Type) *typeInfo {

./html/template/template.go
    func Must(t *Template, err error) *Template {
```

```
./text/template/helper.go
    func Must(t *Template, err error) *Template {

./syscall/dll_windows.go
    func MustLoadDLL(name string) *DLL {

./syscall/dll_windows.go
    func (d *DLL) MustFindProc(name string) *Proc {

./syscall/dll_windows.go
    func (d *LazyDLL) mustLoad() {

./syscall/dll_windows.go
    func (p *LazyProc) mustFind() {

./vendor/golang.org/x/net/dns/dnsmessage/message.go

    func MustNewName(name string) Name {
```

Развернём некоторые из примеров выше:

```
// cmd/internal/bio/must.go
package bio

// MustClose closes Closer c and calls log.Fatal if it returns a
non-nil error.
func MustClose(c io.Closer) {
    if err := c.Close(); err != nil {
        log.Fatal(err)
    }
}
```

```
// cmd/link/internal/ld/target.go
package ld

func (t *Target) mustSetHeadType() {
    if t.HeadType == objabi.Hunknown {
        panic("HeadType is not set")
    }

}
```

```go
// syscall/dll_windows.go
package syscall

// MustLoadDLL is like LoadDLL but panics if load operation fails.
func MustLoadDLL(name string) *DLL {
    d, e := LoadDLL(name)
    if e != nil {
        panic(e)
    }
    return d

}

// fmt/scan.go
package fmt

// mustReadRune turns io.EOF into a panic(io.ErrUnexpectedEOF).
// It is called in cases such as string scanning where an EOF is a
// syntax error.
func (s *ss) mustReadRune() (r rune) {
    r = s.getRune()
    if r == eof {
        s.error(io.ErrUnexpectedEOF)
    }
    return
}

func (s *ss) error(err error) {
    panic(scanError{err})

}

// cmd/internal/bio/must.go
package bio

// MustWriter returns a Writer that wraps the provided Writer,
// except that it calls log.Fatal instead of returning a non-nil
error.
func MustWriter(w io.Writer) io.Writer {
    return mustWriter{w}
}

type mustWriter struct {
    w io.Writer
}
```

```
func (w mustWriter) Write(b []byte) (int, error) {
    n, err := w.w.Write(b)
    if err != nil {
        log.Fatal(err)
    }
    return n, nil
}

func (w mustWriter) WriteString(s string) (int, error) {
    n, err := io.WriteString(w.w, s)
    if err != nil {
        log.Fatal(err)
    }
    return n, nil

}
```

Что лучше, паниковать или фаталить, мы узнаем в следующих уроках. А сейчас главное уловить основную мысль `Must`-функций – **гарантию безошибочного её выполнения** (иными словами "всё или ничего").

Тест "Компиляция Must-функций"

Постарайтесь, не запуская кода, понять, с какими ошибками завершится компиляция следующей программы?

```
1  package main
2
3  import (
4      "fmt"
5      "math/rand"
6      "strconv"
7      "time"
8  )
9
10 func init() {
11         rand.Seed(time.Now().Unix())
12 }
13
14 func main() {
15     for i := 0; i < 3; i++ {
16           fmt.Println(GenerateUniqueID())
17     }
```

```
18 }
19
20 func GenerateUniqueID() string {
21     id := strconv.Itoa(rand.Int())
22     mustNewID(id)
23     knownIDs[id] = struct{}{}
24     return id
25 }
26
27 func mustNewID(id string) {
28     if _, ok := knownIDs[id]; ok {
29         panic("not unique ID: " + id)
30     }
31 }
32
33 var knownIDs = map[string]struct{}{}

34
```

Выберите все подходящие ответы из списка

- [ ] ./main.go:20: function name error: GenerateUniqueID must have "Must" prefix
- [ ] panic: not unique ID: <случайное число>
- [ ] ./main.go:16: implicit usage of must-function without recover()
- [ ] Ни с какими, компиляция пройдёт успешно

В каком месте использовать Must-функции?

Наиболее привычными и часто используемыми `Must`-функциями из стандартной библиотеки Go являются

- template.Must (text)

```
// text/template/helper.go
package template

// Must is a helper that wraps a call to a function returning
(*Template, error)
// and panics if the error is non-nil. It is intended for use in
variable
```

```
// initializations such as
//  var t = template.Must(template.New("name").Parse("text"))
func Must(t *Template, err error) *Template {
    if err != nil {
        panic(err)
    }
    return t

}
```

- template.Must (html)

```
// html/template/template.go
package template

// Must is a helper that wraps a call to a function returning
(*Template, error)
// and panics if the error is non-nil. It is intended for use in
variable initializations
// such as
//  var t = template.Must(template.New("name").Parse("html"))
func Must(t *Template, err error) *Template {
    if err != nil {
        panic(err)
    }
    return t

}
```

- regexp.MustCompile

```
// regexp/regexp.go
package template

// MustCompile is like Compile but panics if the expression cannot be
parsed.
// It simplifies safe initialization of global variables holding
compiled regular
// expressions.
func MustCompile(str string) *Regexp {
    regexp, err := Compile(str)
```

```
    if err != nil {
        panic(`regexp: Compile(` + quote(str) + `): ` + err.Error())
    }
    return regexp

}
```

На их примере покажем, что одним из желательных ~~и дозволенных~~ мест для вызова `Must`-функций является **уровень пакета**.

Как раз из-за того, что данные функции могут паниковать, не следует вызывать их внутри *обычных* функций, работающих с ошибками и возвращающих в свою очередь ошибки.

А при использовании `Must`-функций для инициализации переменных уровня пакета или внутри `init()` мы получаем уже знакомый нам **assertion**.

Пример инициализации шаблона:

```
// cmd/cover/html.go
package main

var htmlTemplate =
template.Must(template.New("html").Funcs(template.FuncMap{
    "colors": colors,
}).Parse(tmplHTML))

// ...

const tmplHTML = `
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
        <title>{{$pkg := .PackageName}}{{if $pkg}}{{$pkg}}: {{end}}Go
Coverage Report</title>
        <style>
            body {
                background: black;
                color: rgb(80, 80, 80);
            }
```

```
            body, pre, #legend span {
                font-family: Menlo, monospace;
                font-weight: bold;
            }
            #topbar {
                background: black;
                position: fixed;
                top: 0; left: 0; right: 0;
                height: 42px;
                border-bottom: 1px solid rgb(80, 80, 80);
            }
...
    })();
    </script>
</html>

`
```

Пример инициализации регулярных выражений:

```
// cmd/vendor/github.com/google/pprof/profile/legacy_profile.go
package profile

import "regexp"

var (
    countStartRE = regexp.MustCompile(`\A(\w+) profile: total \d+\n\z`)
    countRE      = regexp.MustCompile(`\A(\d+) @( 0x[0-9a-f]+)+\n\z`)

    heapHeaderRE = regexp.MustCompile(`heap profile: *(\d+): *(\d+) *\[ *(\d+): *(\d+) *\] *@ *(heap[_a-z0-9]*)/?(\d*)`)
    heapSampleRE = regexp.MustCompile(`(-?\d+): *(-?\d+) *\[ *(\d+): *(\d+) *] @([ x0-9a-f]*)`)

)
```

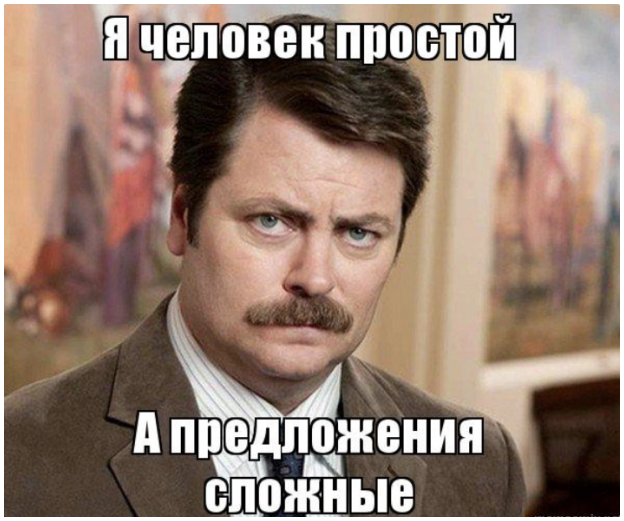
В целом `Must`-функции можно вызывать и на уровне функций/методов, особенно когда это какие-то ваши узкоспециализированные хелперы. Но тогда следует не забывать о риске падения программы и применять те правила, которые мы обсуждали ранее касательно паникования в коде в принципе.

Очевидно, что в тестах рекомендации выше игнорируются и обычно паникуют в любом месте (чтобы поскорее узнать об ошибочной конфигурации теста и сразу же починить её):

```
// https://github.com/golang/go master
$ grep --include "*test*" -E "(m|M)ust\(" -R .
...
./cmd/go/go_test.go
        tg.must(os.MkdirAll(filepath.Join(tg.tempdir,
filepath.Dir(path)), 0755))

...
./cmd/go/internal/test/genflags.go
     t :=
template.Must(template.New("fileTemplate").Parse(fileTemplate))

./cmd/compile/internal/test/testdata/gen/arithConstGen.go
      fncCnst1 := template.Must(template.New("fnc").Parse(

./cmd/compile/internal/test/testdata/gen/arithConstGen.go
      fncCnst2 := template.Must(template.New("fnc").Parse(

...
./cmd/compile/internal/ssa/stmtlines_test.go
               must(err)

./cmd/link/elf_test.go
     tmpl :=
template.Must(template.New("pie").Parse(pieSourceTemplate))

./cmd/nm/nm_test.go
        err =
template.Must(template.New("main").Parse(testexec)).Execute(file,
iscgo)

...
./html/template/clone_test.go
      Must(t3.Parse(`{{define "lhs"}} <style> {{end}}`))

...
./testing/benchmark_test.go
            templ := template.Must(template.New("test").Parse("Hello,
{{.}}!"))

./text/template/examplefiles_test.go
```

```
        tmpl := template.Must(template.ParseGlob(pattern))

...
./text/template/exec_test.go
            tmpl := Must(New("tmpl").Parse(text))
...
./debug/pe/file_test.go

         err =
template.Must(template.New("main").Parse(testprog)).Execute(file,
linktype != linkNoCgo)
```

Так же разработчики часто позволяют себе паниковать в тулзах, имеющих короткий жизненный цикл и направленных на парсинг чего либо, генерацию отчётов, компиляцию и т.п, то есть в местах, которые должны железно работать в общеизвестных, но узкоспециализированных кейсах, а если паника произойдёт, то ничего страшного не случится (в отличие от условного серверного приложения, живущего в облаках и обязанного быть доступным).



Мы с Тамарой ходим парой

Если вы экспортируете из своего пакета `Must`-функцию, то хорошим тоном будет являться предоставление и её менее строгого аналога, не паникующего, а возвращающего ошибку.

Более того, так даже проще и изящнее можно реализовать `Must`-вариант: в нём нужно вызвать оригинальную функцию и при ошибке спаниковать ей, что мы и видим на примере пакета `regexp`:

```go
// regexp/regexp.go
package regexp

func MustCompile(str string) *Regexp {
    regexp, err := Compile(str)
    if err != nil {
        panic(`regexp: Compile(` + quote(str) + `): ` + err.Error())
    }
    return regexp
}

func Compile(expr string) (*Regexp, error) {
    return compile(expr, syntax.Perl, false)
}

func MustCompilePOSIX(str string) *Regexp {
    regexp, err := CompilePOSIX(str)
    if err != nil {
        panic(`regexp: CompilePOSIX(` + quote(str) + `): ` +
err.Error())
    }
    return regexp
}

func CompilePOSIX(expr string) (*Regexp, error) {
    return compile(expr, syntax.POSIX, true)
}
```

Ещё пара примеров из реальных проектов:

```go
package cursor

import (
    "encoding/json"
    // ...
)

type Cursor []byte

func Must(view interface{}) Cursor {
    c, err := New(view)
    if err != nil {
        panic(err)
```

```
    }
    return c
}

func New(view interface{}) (Cursor, error) {
    if helpers.IsNil(view) {
        return nil, nil
    }

    data, err := json.Marshal(view)
    if err != nil {
        return nil, errors.Wrap(err, "marshal view")
    }
    return data, nil

}
```

```
package log

import (
    "context"
    // ...
)

type (
    loggerContextKey  struct{}
    factoryContextKey struct{}
)

func FactoryFromCtx(ctx context.Context) (Factory, bool) {
    f, ok := ctx.Value(factoryContextKey{}).(Factory)
    return f, ok
}

func MustFactoryFromCtx(ctx context.Context) Factory {
    f, ok := FactoryFromCtx(ctx)
    if !ok {
        panic("log factory was not found in ctx")
    }
    return f

}
```

Задача "Must Helper"

Ссылка на заготовку.

Вам необходимо реализовать полиморфную функцию `Must`:

```
// Must выполняет функцию fn и паникует, если та завершилась с ошибкой.
// В обратном случае возвращает результат работы функции.

func Must()
```

Сигнатура скрыта, её вам необходимо додумать таким образом, чтобы можно было работать с функциями, возвращающими значения разных типов:

```
result := Must(func() (*exec.Cmd, error) { // Обратите внимание, что result будет иметь тип *exec.Cmd
    return cmd, nil
})

Must(func() (string, error) {
    return "", errExpected
})


// и т.д.
```

Полезные ссылки:

- go.dev: Tutorial: Getting started with generics
- go.dev: Managing Go installations

- John Arundel: Generics in Go

Больше подробностей см. в заготовке задачи и тестах. Обратите внимание, что тесты менять нельзя.

Задача "MustParseDuration"

Ссылка на заготовку.

1. Сначала вам необходимо реализовать функцию `ParseDuration`, позволяющую распарсить строковое представление *длительности*, лежащее в переменной окружения:

```
func ParseDuration(envName string) (time.Duration, error)
```

2. Затем на основе функции выше необходимо реализовать `MustParseDuration`, паникующую, если распарсить значение не вышло:

```
func MustParseDuration(envName string) time.Duration
```

Пример использования:

```
$ cd tasks/03-panic-concept/must-parse-duration/example
```

```
$ SYNC_PERIOD=3s go run main.go
2022/03/13 21:24:04 sync...
2022/03/13 21:24:07 sync...

2022/03/13 21:24:10 sync...
```

Больше подробностей см. в заготовке задачи и тестах.

P.S. Не нужно писать свою функцию парсинга, в этом вам поможет пакет time!

P.P.S. Обратите внимание на префикс `parse duration: `, добавляемый к ошибке парсинга.

Задача "MustAdapt"

Ссылка на заготовку.

Вы разработчик IoT-платформы, работающей со всякими железяками.

Вам необходимо написать две пары адаптеров из констант сгенерированного контракта

```
type ProtoDeviceType int32

const (
    DeviceType_DEVICE_TYPE_UNSPECIFIED       ProtoDeviceType = 0
    DeviceType_DEVICE_TYPE_ARDUINO_MEGA_2560 ProtoDeviceType = 1
    DeviceType_DEVICE_TYPE_RASPBERRY_PI_4    ProtoDeviceType = 2
    DeviceType_DEVICE_TYPE_LOGI_BONE_2       ProtoDeviceType = 3

)
```

в ваш собственный enum

```
type DeviceType int

const (
    DeviceTypeUnspecified DeviceType = iota
    DeviceTypeArduinoMega2560
    DeviceTypeRaspberryPi4
    DeviceTypeLogiBone2
)
```

и обратно.

Больше подробностей см. в заготовке задачи и тестах.

Задача "Registration Service"

Ссылка на заготовку.

Перед вами основной метод сервиса регистрации пользователей, написанного младшим разработчиком и попавшего в production без ревью (так вот случайно вышло):

```
func (s *Service) SignUp(email, password string) error {
    if err := s.validateEmail(email); err != nil {
        return fmt.Errorf("validate email: %w", err)
    }

    if err := s.validatePassword(password); err != nil {
        return fmt.Errorf("validate password: %w", err)
    }

    // Other sign up code...
    return nil

}
```

Валидацию электронной почты и пароля нового юзверя разработчик решил реализовать через ленивую компиляцию соответствующих регулярок:

```
func (s *Service) validateEmail(email string) error {
    if s.emailRegexp == nil {
        s.emailRegexp = regexp.MustCompile(s.emailExpr)
    }
    // Using of s.emailRegexp
    // ...
}

func (s *Service) validatePassword(password string) error {
    if len(s.passwordRegexps) == 0 {
        s.passwordRegexps = make([]*regexp.Regexp,
len(s.passwordExprs))
        for i, v := range s.passwordExprs {
            s.passwordRegexps[i] = regexp.MustCompile(v)
        }
    }
    // Using of s.passwordRegexps
    // ...
}
```

В какой-то момент приложение упало с паникой (пример):

```
panic: regexp: Compile(`(.*[a-z]{3,}`): error parsing regexp: missing
closing ): `(.*[a-z]{3,}`
```

Оставив целевую платформу на несколько часов без новых пользователей.

Данный инцидент привлёк внимание бизнеса, вся команда получила по шапке, а более опытными товарищами было решено:

- переписать код таким образом, чтобы компиляция невалидной регулярки приводила к возврату ошибки, а не панике;
- избавиться от ленивой компиляции в пользу компиляции в момент конструирования сервиса (читай, на старте приложения);
- исправить прочие ошибки, вылезшие в результате ревью.

Метод `(*Service).SignUp` остаётся без изменений. Остальной код вам необходимо исправить в соответствии с пунктами выше.

Больше подробностей см. в заготовке задачи и тестах.

mustBeNil(err)

Напоследок рассмотрим функцию, которую в различных формах можно встретить в проектах, написанных на Go:

```
func must(err error) { // mustNil, mustBeNil, mustNoErr, etc.
    if err != nil {
        panic(err)
    }

}
```

Данный хелпер позволяет сократить код, связанный с обработкой ошибок, и особенно полезен в тестах, CLI-тулзах, примерах и т.д. (пример использования из первого модуля нашего курса).

В стандартной библиотеке подобные функции тоже встречаются:

```
// cmd/go/go_test.go
```

```go
package main_test

// must gives a fatal error if err is not nil.
func (tg *testgoData) must(err error) {
    tg.t.Helper()
    if err != nil {
        tg.t.Fatal(err)
    }
}



// cmd/compile/internal/ssa/stmtlines_test.go
package ssa_test

func must(err error) {
    if err != nil {
        panic(err)
    }

}

// testing/cover.go
package testing

// mustBeNil checks the error and, if present, reports it and exits.
func mustBeNil(err error) {
    if err != nil {
        fmt.Fprintf(os.Stderr, "testing: %s\n", err)
        os.Exit(2)
    }

}
```

Более продвинутый аналог был предложен в [golang/go: proposal: Go 2: builtin: must function](), но proposal принят не был.

Несмотря на это **Go 1.18** позволяет [реализовать]() предложенную выше встроенную функцию собственными силами:

```go
func Must[T any](v T, err error) T {
    if err != nil {
        panic(err.Error())
```

```
    }
    return v
}

func main() {
    f := Must(os.Create("test.txt"))
    // ...

}
```

Мотаем на ус, пользуемся!

Выводы

В сухом остатке мы узнали, что:

- Если видите префикс `Must`/`must`, то будьте готовы, что функция "бахает"!
- Как следствие старайтесь сами придерживаться данной конвенции при создании своих "опасных" функций.
- Экспортируя `Must`-функцию, будьте добры предоставить её "безопасный" аналог.
- Уровень пакета или тесты – прекрасное место для использования `Must`-функций.
- Не стесняйтесь создавать хелперы, делающие ваш код более лаконичным.