Go Generics: First Impressions

@nikolaydubina
2022-04-27

Validation with go-playground

github.com/go-playground/validator

- Define as struct tags
- Initialize *Validator* instance
- *Validator* constructed with reflection
- Very fast 100s of ns

```go
// User contains user information
type User struct {
	FirstName      string     `validate:"required"`
	LastName       string     `validate:"required"`
	Age            uint8      `validate:"gte=0,lte=130"`
	Email          string     `validate:"required,email"`
	FavouriteColor string     `validate:"iscolor"`                // alias for 'hexcolor|rgb|rgba|hsl|hsla'
	Addresses      []*Address `validate:"required,dive,required"` // a person can have a home and cottage...
}

// Address houses a users address information
type Address struct {
	Street string `validate:"required"`
	City   string `validate:"required"`
	Planet string `validate:"required"`
	Phone  string `validate:"required"`
}

// use a single instance of Validate, it caches struct info
var validate *validator.Validate

func main() {

	validate = validator.New()

	validateStruct()
	validateVariable()
}
```

Validation with native Go constructs

github.com/nikolaydubina/validate

- Fast and small LOC validation
- No reflection
- No gencode
- No tags
- Composable
- Short notation
- How convenient, small LOC, and fast can it be?

```go
// Employee is example of struct with validatable fields and nested structure
type Employee struct {
	Name          string
	Age           int
	Color         Color     // custom func Validate()
	Education     Education // nested with Validate()
	Salary        float64
	Experience    time.Duration
	Birthday      time.Time
	VacationStart time.Time
}

func (s Employee) Validate() error {
	return validate.All(
		validate.OneOf("name", s.Name, "Zeus", "Hera"),
		validate.OneOf("age", s.Age, 35, 55),
		validate.Min("age", s.Age, 10), // same field validated again
		s.Color.Validate(),
		s.Education.Validate(),
		validate.Max("salary", s.Salary, 123.456),
		validate.Max("duration", s.Experience, time.Duration(1)*time.Hour),
		validate.After("birthday", s.Birthday, time.Date(1984, 1, 1, 0, 0, 0, 0, time.UTC)),
		validate.Before("vacation_start", s.VacationStart, time.Date(2024, 1, 1, 0, 0, 0, 0
	)
}

// Education is another custom struct
type Education struct {
	Duration   int
	SchoolName string
}

func (e Education) Validate() error {
	if (e.Duration % 17) == 5 {
		return errors.New("my special error")
	}
	return validate.All(
		validate.Min("", e.Duration, 10),
		validate.OneOf("", e.SchoolName, "KAIST", "Stanford"),
	)
}
```

Learnings: Go generics work well

.. and barely noticeable

```go
func (s Employee) Validate() error {
        return validate.All(
                validate.OneOf("name", s.Name, "Zeus", "Hera"),
                validate.OneOf("age", s.Age, 35, 55),
                validate.Min("age", s.Age, 10), // same field val
                s.Color.Validate(),
```

```go
type errOneOf[T any] struct {
        Name    string
        Val     T
        Options []T
}

func (e errOneOf[T]) Error() string {
        return fmt.Sprintf("%s(%v) not in %v", e.Name, e.Val, e.Options)
}

func OneOf[T comparable](name string, v T, options ...T) error {
        for _, q := range options {
                if v == q {
                        return nil
                }
        }
        return errOneOf[T]{Name: name, Val: v, Options: options}
}
```

Learnings: Go generics have many limitations

- Can not convert T1 to T2

No way to express convertibility

The design has no way to express convertibility between two different type parameters. For example, there is no way to write this function:

```
// Copy copies values from src to dst, converting them as they go.
// It returns the number of items copied, which is the minimum of
// the lengths of dst and src.
// This implementation is INVALID.
func Copy[T1, T2 any](dst []T1, src []T2) int {
        for i, x := range src {
                if i > len(dst) {
                        return i
                }
                dst[i] = T1(x) // INVALID
        }
        return len(src)
}
```

The conversion from type `T2` to type `T1` is invalid, as there is no constraint on either type that permits the conversion. Worse, there is no way to write such a constraint in general. In the particular case where both `T1` and `T2` have limited type sets this function can be written as described earlier when discussing type conversions using type sets. But, for example, there is no way to write a constraint for the case in which `T1` is an interface type and `T2` is a type that implements that interface.

It's worth noting that if `T1` is an interface type then this can be written using a conversion to the empty interface type and a type assertion, but this is, of course, not compile-time type-safe.

```
// Copy copies values from src to dst, converting them as they go.
// It returns the number of items copied, which is the minimum of
// the lengths of dst and src.
func Copy[T1, T2 any](dst []T1, src []T2) int {
        for i, x := range src {
                if i > len(dst) {
                        return i
                }
                dst[i] = (interface{})(x).(T1)
        }
        return len(src)
}
```

Ref. https://go.googlesource.com/proposal/+/refs/heads/master/design/43651-type-parameters.md#omissions

Learnings: Go generics have many limitations

- Can not convert T1 to T2
- Can not identity matched replaced type

Identifying the matched predeclared type

The design doesn't provide any way to test the underlying type matched by a `~T` constraint element. Code can test the actual type argument through the somewhat awkward approach of converting to an empty interface type and using a type assertion or a type switch. But that lets code test the actual type argument, which is not the same as the underlying type.

Here is an example that shows the difference.

```
type Float interface {
        ~float32 | ~float64
}

func NewtonSqrt[T Float](v T) T {
        var iterations int
        switch (interface{})(v).(type) {
        case float32:
                iterations = 4
        case float64:
                iterations = 5
        default:
                panic(fmt.Sprintf("unexpected type %T", v))
        }
        // Code omitted.
}

type MyFloat float32

var G = NewtonSqrt(MyFloat(64))
```

This code will panic when initializing `G`, because the type of `v` in the `NewtonSqrt` function will be `MyFloat`, not `float32` or `float64`. What this function actually wants to test is not the type of `v`, but the approximate type that `v` matched in the constraint's type set.

One way to handle this would be to permit writing approximate types in a type switch, as in `case ~float32:`. Such a case would match any type whose underlying type is `float32`. This would be meaningful, and potentially useful, even in type switches outside of generic functions.

Ref. https://go.googlesource.com/proposal/+/refs/heads/master/design/43651-type-parameters.md#omissions

Learnings: Go generics have many limitations

- Can not convert T1 to T2
- Can not identity matched replaced type
- T can not be slice, nor array, not map
  - Can not iterate over it
  - Can not access at index

```
1  package main
2
3  import (
4          "fmt"
5  )
6
7  func Print[T any](s T) {
8          for _, v := range s {
9                  fmt.Print(v)
10         }
11 }
12
13 func main() {
14         x := map[stirng]int{"a": 1}
15         Print(x)
16 }
17
```

```
./prog.go:8:20: cannot range over s (variable of type T constrained by any) (T has no core type)
./prog.go:14:11: undefined: stirng

Go build failed.
```

Learnings: Go generics have many limitations

- Can not convert T1 to T2
- Can not identity matched replaced type
- T can not be slice, nor array, not map
- ..and many more

Omissions

We believe that this design covers the basic requirements for generic programming. However, there are a number of programming constructs that are not supported.

- No specialization. There is no way to write multiple versions of a generic function that are designed to work with specific type arguments.
- No metaprogramming. There is no way to write code that is executed at compile time to generate code to be executed at run time.
- No higher level abstraction. There is no way to use a function with type arguments other than to call it or instantiate it. There is no way to use a generic type other than to instantiate it.
- No general type description. In order to use operators in a generic function, constraints list specific types, rather than describing the characteristics that a type must have. This is easy to understand but may be limiting at times.
- No covariance or contravariance of function parameters.
- No operator methods. You can write a generic container that is compile-time type-safe, but you can only access it with ordinary methods, not with syntax like `c[k]`.
- No currying. There is no way to partially instantiate a generic function or type, other than by using a helper function or a wrapper type. All type arguments must be either explicitly passed or inferred at instantiation time.
- No variadic type parameters. There is no support for variadic type parameters, which would permit writing a single generic function that takes different numbers of both type parameters and regular parameters.
- No adaptors. There is no way for a constraint to define adaptors that could be used to support type arguments that do not already implement the constraint, such as, for example, defining an `==` operator in terms of an `Equal` method, or vice-versa.
- No parameterization on non-type values such as constants. This arises most obviously for arrays, where it might sometimes be convenient to write `type Matrix[n int] [n][n]float64`. It might also sometimes be useful to specify significant values for a container type, such as a default value for elements.

Ref. https://go.googlesource.com/proposal/+/refs/heads/master/design/43651-type-parameters.md#omissions

Learnings: wrapping functions slows down

Time increases and mallocs too

```go
func (s Employee) Validate() error {
        return validate.All(
                validate.OneOf[string]{Name: "name", Value: s.Name, Values: []string{"Zeus", "Hera"}
                validate.OneOf[int]{Name: "age", Value: s.Age, Values: []int{35, 55}},
                validate.Min[int]{Name: "age", Value: s.Age, Min: 10}, // same field validated agai
                s.Color,
                s.Education,
                validate.Max[float64]{Name: "salary", Value: s.Salary, Max: 123.456},
                validate.Max[time.Duration]{Name: "duration", Value: s.Experience, Max: time.Duratio
                validate.After{Name: "birthday", Value: s.Birthday, Time: time.Date(1984, 1, 1, 0, 0
                validate.Before{Name: "vacation_start", Value: s.VacationStart, Time: time.Date(2024
        )
}
```

```
$ go test -bench=. -benchtime=10s -benchmem ./...
goos: darwin
goarch: amd64
pkg: github.com/nikolaydubina/validate
cpu: VirtualApple @ 2.50GHz
BenchmarkEmployee_Error_Message-10                        3579223              3379 ns/op            2761 B/op         62 allocs/op
BenchmarkEmployee_Error-10                                9361948              1277 ns/op            1344 B/op         34 allocs/op
BenchmarkEmployee_Success-10                             25418672               474 ns/op             552 B/op         14 allocs/op
BenchmarkEmployeeSimple_Error_Message-10                  8757170              1364 ns/op             992 B/op         28 allocs/op
BenchmarkEmployeeSimple_Error-10                         30418941               394 ns/op             504 B/op         13 allocs/op
BenchmarkEmployeeSimple_Success-10                       65194581               184 ns/op             224 B/op          6 allocs/op
BenchmarkEmployeeNoContainers_Error_Message-10           24971338               483 ns/op             280 B/op         10 allocs/op
BenchmarkEmployeeNoContainers_Error-10                   72736639               165 ns/op             136 B/op          5 allocs/op
BenchmarkEmployeeNoContainers_Success-10                143333276                83 ns/op              64 B/op          2 allocs/op
PASS
ok      github.com/nikolaydubina/validate       124.950s
```

```
$ go test -bench=. -benchtime=10s -benchmem ./...
goos: darwin
goarch: amd64
pkg: github.com/nikolaydubina/validate
cpu: VirtualApple @ 2.50GHz
BenchmarkEmployee_Error_Message-10                        3744121              3229 ns/op            2376 B/op         56 allocs/op
BenchmarkEmployee_Error-10                               12533948               958 ns/op             904 B/op         23 allocs/op
BenchmarkEmployee_Success-10                            100000000               115 ns/op              80 B/op          3 allocs/op
BenchmarkEmployeeSimple_Error_Message-10                  9488436              1263 ns/op             840 B/op         25 allocs/op
BenchmarkEmployeeSimple_Error-10                         44261380               270 ns/op             344 B/op          9 allocs/op
BenchmarkEmployeeSimple_Success-10                      243491635                49 ns/op              48 B/op          2 allocs/op
BenchmarkEmployeeNoContainers_Error_Message-10           28089966               427 ns/op             248 B/op          9 allocs/op
BenchmarkEmployeeNoContainers_Error-10                  142881793                85 ns/op              88 B/op          3 allocs/op
BenchmarkEmployeeNoContainers_Success-10               1000000000                 4 ns/op               0 B/op          0 allocs/op
PASS
ok      github.com/nikolaydubina/validate       120.565s
```

Learnings: passing functions through maps slows down

Overhead of indirection and map is higher than saving from strings

```go
func (s Employee) Validate() error {
	return validate.All(map[string]error{
		"name":           validate.OneOf(s.Name, "Zeus", "Hera"),
		"age":            validate.OneOf(s.Age, 35, 55),
		"age_2":          validate.Min(s.Age, 10), // same field validated again
		"color":          s.Color.Validate(),
		"education":      s.Education.Validate(),
		"salary":         validate.Max(s.Salary, 123.456),
		"duration":       validate.Max(s.Experience, time.Duration(1)*time.Hour),
		"birthday":       validate.After(s.Birthday, time.Date(1984, 1, 1, 0, 0, 0, 0, time.UTC)),
		"vacation_start": validate.Before(s.VacationStart, time.Date(2024, 1, 1, 0, 0, 0, 0, time.UTC)),
	})
}
```

```
goos: darwin
goarch: amd64
pkg: github.com/nikolaydubina/validate
cpu: VirtualApple @ 2.50GHz
BenchmarkEmployee_Error_Message-10                      2698065              4359 ns/op            3866 B/op         57 allocs/op
BenchmarkEmployee_Error-10                              6993564              1714 ns/op            2058 B/op         21 allocs/op
BenchmarkEmployee_Success-10                           14948445               810 ns/op            1329 B/op          7 allocs/op
BenchmarkEmployeeSimple_Error_Message-10                8243392              1460 ns/op            1112 B/op         26 allocs/op
BenchmarkEmployeeSimple_Error-10                       33798837               356 ns/op             496 B/op          7 allocs/op
BenchmarkEmployeeSimple_Success-10                     66953932               182 ns/op              96 B/op          3 allocs/op
BenchmarkEmployeeNoContainers_Error_Message-10         19754359               600 ns/op             576 B/op         10 allocs/op
BenchmarkEmployeeNoContainers_Error-10                 61285670               194 ns/op             368 B/op          3 allocs/op
BenchmarkEmployeeNoContainers_Success-10               123293473               98 ns/op              48 B/op          1 allocs/op
PASS
ok      github.com/nikolaydubina/validate       128.263s
```

```
$ go test -bench=. -benchtime=10s -benchmem ./...
goos: darwin
goarch: amd64
pkg: github.com/nikolaydubina/validate
cpu: VirtualApple @ 2.50GHz
BenchmarkEmployee_Error_Message-10                      3744121              3229 ns/op            2376 B/op         56 allocs/op
BenchmarkEmployee_Error-10                             12533948               958 ns/op             904 B/op         23 allocs/op
BenchmarkEmployee_Success-10                          100000000               115 ns/op              80 B/op          3 allocs/op
BenchmarkEmployeeSimple_Error_Message-10                9488436              1263 ns/op             840 B/op         25 allocs/op
BenchmarkEmployeeSimple_Error-10                       44261380               270 ns/op             344 B/op          9 allocs/op
BenchmarkEmployeeSimple_Success-10                    243491635                49 ns/op              48 B/op          2 allocs/op
BenchmarkEmployeeNoContainers_Error_Message-10         28089966               427 ns/op             248 B/op          9 allocs/op
BenchmarkEmployeeNoContainers_Error-10                 142881793               85 ns/op              88 B/op          3 allocs/op
BenchmarkEmployeeNoContainers_Success-10              1000000000                4 ns/op               0 B/op          0 allocs/op
PASS
ok      github.com/nikolaydubina/validate       120.565s
```

Learnings: fmt is slow + lazy execution

- Hot-path: error does not happen
  - Avoid mallocs when error not happens
  - Terminate ASAP
- Lazy execution in Go is possible
- fmt is expensive
  - create container needed for rendering
  - invoke render only to when Error() invoked
  - try to render with non-fmt first errors.New is fast

```go
type errOneOf[T any] struct {
	Name    string
	Val     T
	Options []T
}

func (e errOneOf[T]) Error() string {
	return fmt.Sprintf("%s(%v) not in %v", e.Name, e.Val, e.Options)
}

func OneOf[T comparable](name string, v T, options ...T) error {
	for _, q := range options {
		if v == q {
			return nil
		}
	}
	return errOneOf[T]{Name: name, Val: v, Options: options}
}
```

```
$ go test -bench=. -benchtime=10s -benchmem ./...
goos: darwin
goarch: amd64
pkg: github.com/nikolaydubina/validate
cpu: VirtualApple @ 2.50GHz
BenchmarkEmployee_Error_Message-10                        3744121          3229 ns/op            2376 B/op         56 allocs/op
BenchmarkEmployee_Error-10                               12533948           958 ns/op             904 B/op         23 allocs/op
BenchmarkEmployee_Success-10                            100000000           115 ns/op              80 B/op          3 allocs/op
BenchmarkEmployeeSimple_Error_Message-10                  9488436          1263 ns/op             840 B/op         25 allocs/op
BenchmarkEmployeeSimple_Error-10                         44261380           270 ns/op             344 B/op          9 allocs/op
BenchmarkEmployeeSimple_Success-10                      243491635            49 ns/op              48 B/op          2 allocs/op
BenchmarkEmployeeNoContainers_Error_Message-10           28089966           427 ns/op             248 B/op          9 allocs/op
BenchmarkEmployeeNoContainers_Error-10                  142881793            85 ns/op              88 B/op          3 allocs/op
BenchmarkEmployeeNoContainers_Success-10               1000000000             4 ns/op               0 B/op          0 allocs/op
PASS
ok      github.com/nikolaydubina/validate       120.565s
```

More benchmarking errors
https://github.com/nikolaydubina/go-bench-errors

Final outcome?

- Used generics only for function parameters
- Functions instead of interfaces
- Fairly fast
- ~90LOC ok syntax
- Generics do reduce duplication for basic types
- Generics are not as flexible

Official Go recommendation

“..should avoid type parameters until you notice that you are about the write the exact same code multiple times.”

- Don’t replace interface types with type parameters
- Don’t use type parameters if method implementations differ

Ref. https://go.dev/blog/when-generics

Other mechanisms: go:generate and tags

https://github.com/nikolaydubina/go-featureprocessing

- Generate in-place indexing for invocations
- For speed + json tags

```
//go:generate go run github.com/nikolaydubina/go-featureprocessing/cmd/generate -struct=Employee

type Employee struct {
        Age         int     `feature:"identity"`
        Salary      float64 `feature:"minmax"`
        Kids        int     `feature:"maxabs"`
        Weight      float64 `feature:"standard"`
        Height      float64 `feature:"quantile"`
        City        string  `feature:"onehot"`
        Car         string  `feature:"ordinal"`
        Income      float64 `feature:"kbins"`
        Description string  `feature:"tfidf"`
        SecretValue float64
}
```

```
// TransformInplace transforms struct into feature vector accordingly to transformers, and does so inplace
func (e *AllTransformersFeatureTransformer) TransformInplace(dst []float64, s *AllTransformers) {
        if s == nil || e == nil || len(dst) != e.NumFeatures() {
                return
        }
        idx := 0

        dst[idx] = e.Name0.Transform(float64(s.Name0))
        idx++

        dst[idx] = e.Name1.Transform(float64(s.Name1))
        idx++

        dst[idx] = e.Name2.Transform(float64(s.Name2))
        idx++

        dst[idx] = e.Name3.Transform(float64(s.Name3))
        idx++

        dst[idx] = e.Name4.Transform(float64(s.Name4))
        idx++

        e.Name5.TransformInplace(dst[idx:idx+e.Name5.NumFeatures()], s.Name5)
        idx += e.Name5.NumFeatures()

        dst[idx] = e.Name6.Transform((s.Name6))
        idx++
```

```
// FeatureNames provides names of features that match output of transform
func (e *AllTransformersFeatureTransformer) FeatureNames() []string {
        if e == nil {
                return nil
        }

        idx := 0
        names := make([]string, e.NumFeatures())

        names[idx] = "Name0"
        idx++

        names[idx] = "Name1"
        idx++

        names[idx] = "Name2"
        idx++

        names[idx] = "Name3"
        idx++

        names[idx] = "Name4"
        idx++

        for _, w := range e.Name5.FeatureNames() {
                names[idx] = "Name5_" + w
                idx++
        }

        names[idx] = "Name6"
        idx++

        names[idx] = "Name7"
        idx++

        for _, w := range e.Name8.FeatureNames() {
                names[idx] = "Name8_" + w
                idx++
        }

        for _, w := range e.Name9.FeatureNames() {
                names[idx] = "Name9_" + w
                idx++
        }

        return names
}
```

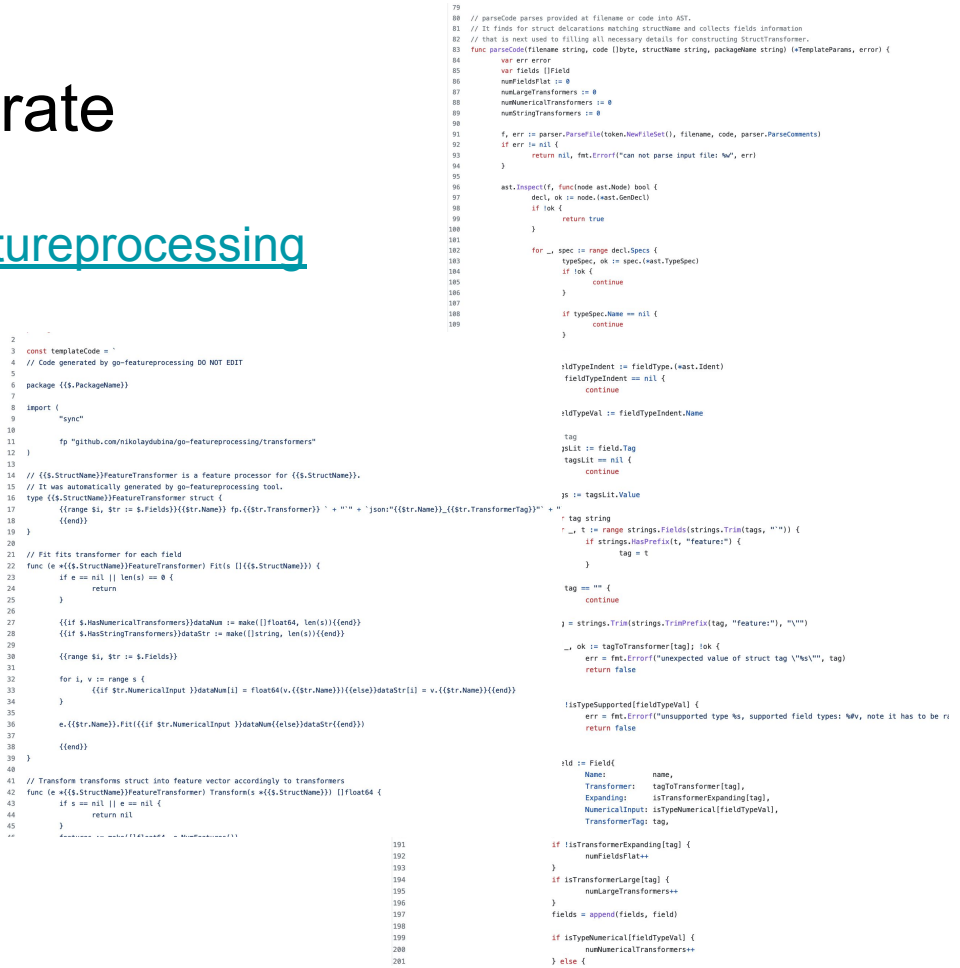
Other mechanisms: go:generate

https://github.com/nikolaydubina/go-featureprocessing

- Very fast
- Functions get inlined

However!

- Laborious to write AST generator



Thank you!