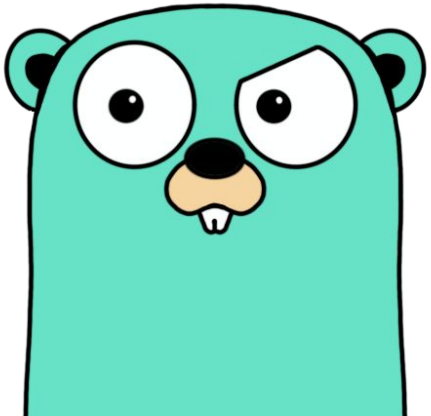
Generics in Go (benchmarks)



Автор: Денис Лимарев

Введение

Версия Go - 1.18.1beta

Процессор: 2,3 GHz 8-ядерный процессор Intel Core i9, RAM:16 ГБ DDR4

Архитектура/ОС - darwin/amd64, macOS Catalina 10.15.7

Материалы доклада: https://github.com/peakle/meetup/tree/master/go-generics

Цитата из спецификации

Generic functions, rather than generic types, can probably be compiled using an interface-based approach. That will optimize compile time, in that the function is only compiled once, but there will be some run time cost.

Generic types may most naturally be compiled multiple times for each set of type arguments. This will clearly carry a compile time cost, but there shouldn't be any run time cost. Compilers can also choose to implement generic types similarly to interface types, using special purpose methods to access each element that depends on a type parameter.

```go
type StackGenerics[T any] []T

type StackInterface []interface{}

type StackFoo []Foo
```

```
105  func BenchmarkStackGenericsBar(b *testing.B) {
106      var s StackGenerics[Bar]
107      for i := 0; i < b.N; i++ {
108          s.Push(Bar{})
109          s.Push(Bar{})
110          s.Pop()
111          bar = s.Peek()
112      }
```

```go
////// Methods for StackGenerics type ///////////////
func (s StackGenerics[T]) Peek() T {
    return s[len(s)-1]
}

func (s *StackGenerics[T]) Pop() {
    *s = (*s)[:len(*s)-1]
}

func (s *StackGenerics[T]) Push(value T) {
    *s = append(*s, value)
}

```

```
55  ////// Methods for StackFoo type ////////////////
56  func (s StackFoo) Peek() Foo {
57      return s[len(s)-1]
58  }
59
60  func (s *StackFoo) Pop() {
61      *s = (*s)[:len(*s)-1]
62  }
63
64  func (s *StackFoo) Push(value Foo) {
65      *s = append(*s, value)
66  }
67
```

```go
////// Generic function ////////////////////

func Peek[T any](s []T) T {
    return s[len(s)-1]
}

func  Pop[T any](s []T) []T{
    s = s[:len(s)-1]
    return s
}

func  Push[T any](s []T, value T)[]T {
    s = append(s, value)
    return s
}
```

Сравнение подходов типизации

Название	Время на операцию (ns/op)	Память (Мб)	Циклов GC	Количество запусков в бенчмарке
interface{} тип	57.20	9192	35	100 000 000
конкретный тип	10.38	4376	26	100 000 000
generics тип	10.38	4376	24	100 000 000
generics функция	7.16	4376	26	100 000 000

Синтаксическая сложность

```
type StackGenerics[T any] []T

type StackInterface []interface{}

type StackFoo []Foo
```

```go
func (s StackGenerics[T]) Peek() T {
    return s[len(s)-1]
}
```

```go
var s StackGenerics[Foo]
for i := 0; i < b.N; i++ {
    s.Push(Foo{})
```

Выводы

Плюсы:

- Уменьшают размер кодовой базы
- Упрощают поддержку кода
- Убирают необходимость использования interface{}
- Уменьшают нагрузку на GC (в случае смены с interface{} типа)

Минусы:

- Повышенная синтаксическая сложность
- Более долгое время компиляции программы
- Временное отсутствие поддержки в IDE