Урок Спринт 1.10

Type Constraint

Type Constraint в Go полезен для того, чтобы ограничить типы, которые могут быть использованы в качестве параметров дженериков.

В Go есть несколько встроенных типов данных, в том числе `Complex`, `Float`, `Integer`, `Ordered`, `Signed` и `Unsigned`:

- Тип `Integer` — это целочисленные значения. Может быть представлен в виде `int`, `int8`, `int16`, `int32` или `int64`, в зависимости от выбранной точности
- Тип `Ordered` — это упорядоченные значения. Может быть представлен в виде `byte`, `rune`, `int`, `int16`, `int32`, `int64`, `float32` или `float64`
- Тип `Complex` — это комплексное число из действительной и мнимой частей. Может быть представлен в виде `complex64` или `complex128`, в зависимости от выбранной точности

Также мы можем ограничивать типы параметров нашего дженерика:

```
type MyConstraint interface {
  int | int8 | int16 | int32 | int64
}
```

Если мы хотим создать дженерик, который принимает только целочисленные параметры, мы можем определить его так:

```
func MyGeneric[T MyConstraint](x T) {
  // ...
}
```

Вот как сработают эти ограничения:

```
type MyConstraint interface {
    int | int8 | int16 | int32 | int64
}

func MyFunc[T MyConstraint](m T) {
    // ...
}

func main() {
    // Получится, потому что тип int входит в список ограничений
    // (int | int8 | int16 | int32 | int64)
    MyFunc[int](1)

    // Не получится, потому что string  — ни один из типов интерфейса MyConstraint
    MyFunc[string]("hello")
}
```

Так же в качестве ограничения может быть:

- `comparable` — любой тип, который можно сравнивать с собой же. Это может быть любой тип или структура в Go, что не содержит функций, слайсов и мап (однако массивы под слайсом `comparable`).
- `[]int` — кроме типа можно задать ограничение как на слайс конкретного типа.
- `~int` — так ограничение будет соответствовать ещё и встроенным типам, например `type MyInt int`.
- `~[]int` — всё это можно комбинировать

Ограничение можно использовать внутри структуры:

```
type List[T any] struct {
    next  *List[T]
    value T
}
```

Так же, ограничение можно использовать для интерфейсов:

```
type MyInt interface {
    ~int
    String() string
} // для соответствия интерфейсу тип должен быть int или встроенным типом к int, и иметь функцию String()
```