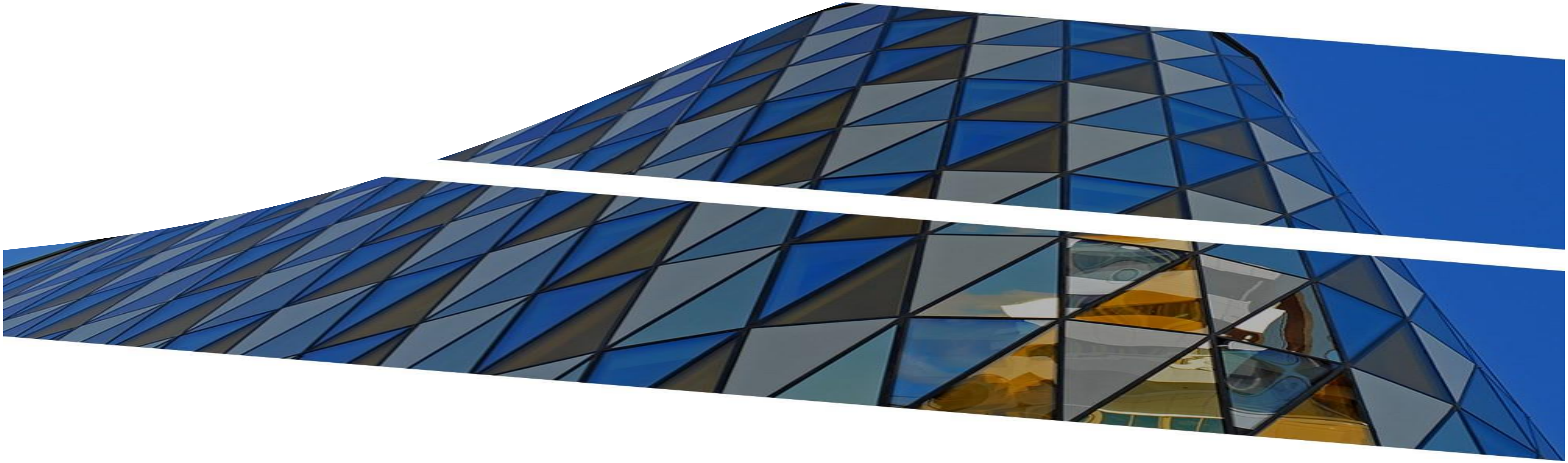

Golang Programming

Using Code Generation to Implement Parametric Types in Go

Where to Find The Code and Materials?

https://github.com/iproduct/coursego

Agenda for This Session

- The problem
- Code generation
- Generics and their purpose
- Type-specific wrappers with type-agnostic implementations
- Alternatives – using interfaces, assertions, reflection, code generation
- Generating code with go generate
- Q&A. Projects mentoring.

The Problem

- Many algorithms and data structures are applicable to many data types. Examples: Stringer, Sorter, priority queue, sorted tree, concurrent hash map
- Container data types above could hold elements of string, int64, []int, map[string]string, or a struct type.
- We could implement a sorting algorithm using sorted tree for example. The requirement is that the elemets should be comparable to each other.
- But what if we want to reuse the same sorted tree implementation with different element types?
- If there are N methods and M element types we should write N x M function implementations …

Solutions from Other Languages – e.g. Java

```java
public class SortedTree<E extends Comparable<E>> {
    public void insert(E element) {
        //...
    }
}


SortedTree<String> stringSortedTree = new SortedTree<>();
sortedTreeOfStrings.insert("xyz");
sortedTreeOfStrings.insert("abc");
sortedTreeOfStrings.insert("klm");
```

The Ways to Implement Generic Code in Go

- “If C++ and Java are about type hierarchies and the taxonomy of types, Go is about composition.” (Rob Pike)
- Use multiple functions (copy & paste ☺)
- Interfaces
- Type assertions
- Reflection
- Code generation

Interfaces (1)

- Example: sort.Interface

```
type Interface interface {
      Len() int
      Less(i, j int) bool
      Swap(i, j int)
}
```

- Example: `io.Reader` interface:
  - many functions take an io.Reader as input
  - many data types, including files, network connections, ciphers, implement it
- Or even: `interface {}. E.g.:`

```
 func Slice(slice interface{}, less func(i, j int) bool)
```

Interfaces (2)

```
import "sort"

type Person struct {
        Name string
        Age  int
}

// ByAge implements sort.Interface for []Person based on the Age field.
type ByAge []Person

func (a ByAge) Len() int           { return len(a) }
func (a ByAge) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
func (a ByAge) Less(i, j int) bool { return a[i].Age < a[j].Age }

func SortPeople(people []Person) {
        sort.Sort(ByAge(people))
}
```

Interfaces (3)

```
package main
import ("fmt"; "sort")
func main() {
        people := []struct {
                Name string
                Age  int
        }{
                {"Gopher", 7},
                {"Alice", 55},
                {"Vera", 24},
                {"Bob", 75},
        }
        sort.Slice(people, func(i, j int) bool { return people[i].Name < people[j].Name })
        fmt.Println("By name:", people)

        sort.Slice(people, func(i, j int) bool { return people[i].Age < people[j].Age })
        fmt.Println("By age:", people)
}
```

Type Assertions (1)

```
type MyContainer []interface{}

func (c *MyContainer) Put(elem interface{}) {
	*c = append(*c, elem)
}

func (c *MyContainer) Get() interface{} {
	elem := (*c)[0]
	*c = (*c)[1:]
	return elem
}
```

Type Assertions (2)

```go
func main() {
    myIntContainer := &MyContainer{}
    myIntContainer.Put(12)
    myIntContainer.Put(70)
    elem, ok := myIntContainer.Get().(int)
    if !ok {
        fmt.Println("Error getting int from myIntContainer")
    }
    fmt.Printf("Got: %d (%T)\n", elem, elem)
}
```

Reflection (1)

```go
package main

import (
        "fmt"
        "reflect"
)

type MyStack struct {
        t reflect.Type
        value reflect.Value
}

func New(tp reflect.Type) *MyStack {
        return &MyStack{
                t: tp,
                value: reflect.MakeSlice(reflect.SliceOf(tp), 0, 100),
        }
}
```

Reflection (2)

```go
func (m *MyStack) Push(v interface{}) {
	if reflect.ValueOf(v).Type() != m.value.Type().Elem() {
		panic(fmt.Sprintf("Error putting %T into %s", v, m.value.Type().Elem()))
	}
	m.value = reflect.Append(m.value, reflect.ValueOf(v))
}

func (m *MyStack) Pop() interface{} {
	v := m.value.Index(0)
	m.value = m.value.Slice(1, m.value.Len())
	return v.Interface()
}

func main() {
	val := 2.88
	stack := New(reflect.TypeOf(val))
	stack.Push(val)
	fmt.Println(stack.value.Index(0))
	result := stack.Pop()
	fmt.Printf("Result: %[1]f (%[1]T)\n", result)
}
```

Code Generation - Rob Pike
[https://blog.golang.org/generate]

- A property of universal computation—Turing completeness—is that a computer program can write a computer program. This is a powerful idea that is not appreciated as often as it might be, even though it happens frequently.

- It's a big part of the definition of a compiler, for instance. It's also how the go test command works: it scans the packages to be tested, writes out a Go program containing a test harness customized for the package, and then compiles and runs it.

- Modern computers are so fast this expensive-sounding sequence can complete in a fraction of a second.

- Examples: Yacc, Protocol Buffers

Code Generation Example - Rob Pike [https://blog.golang.org/generate]

- Get Go Yacc tool:

```
go get golang.org/x/tools/cmd/goyacc
```

- You can run it directly:

```
goyacc -o gopher.go -p parser gopher.y
```

- Go generate - add this comment anywhere in the non-generated go file:

```
//go:generate goyacc -o gopher.go -p parser gopher.y
```

```
$ cd $GOPATH/myrepo/gopher
$ go generate
$ go build
$ go test
```

Code Generation - Rob Pike

[https://blog.golang.org/generate]

- A property of universal computation—Turing completeness—is that a computer program can write a computer program. This is a powerful idea that is not appreciated as often as it might be, even though it happens frequently.

- It's a big part of the definition of a compiler, for instance. It's also how the go test command works: it scans the packages to be tested, writes out a Go program containing a test harness customized for the package, and then compiles and runs it.

- Modern computers are so fast this expensive-sounding sequence can complete in a fraction of a second.

- Examples: Yacc, Protocol Buffers

Code Generation Examples

https://github.com/iproduct/coursego/tree/master/generation
https://github.com/iproduct/coursego/tree/master/generationlab

Q&A. Projects mentoring

Recommended Literature

- The Go Documentation - https://golang.org/doc/
- The Go Bible: Effective Go - https://golang.org/doc/effective_go.html
- David Chisnall, *The Go Programming Language Phrasebook*, Addison Wesley, 2012
- Alan A. A. Donovan, Brian W. Kernighan, *The Go Programming Language*, Addison Wesley, 2016
- Nathan Youngman, Roger Peppé, *Get Programming with Go*, Manning, 2018
- Naren Yellavula, *Building RESTful Web Services with Go*, Packt, 2017

Thank’s for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD