Go 语言从入门到项目实战

第5章 函数

5.1 声明

- 格式：

```
func function_name(params)(return_types){
//要执行的代码块（函数体）
}
```

5.1 声明

- 示例：

```
/*三酷猫打招呼
 */
func main() {
        sayHello()
}
func sayHello(){
        fmt.Println("你好，三酷猫")
}
```

5.1 声明

- 示例：

```
/*三酷猫算加法
 */
func main() {
        calcSum(1.2,3.4)
}
func calcSum(numX,numY float64){
        fmt.Println(numX+numY)
}
```

5.1 声明

- 示例：

```
func main() {
        printStrings("文本",123,false)
}
func printStrings(args...interface{}){
        fmt.Println(args)
}
```

5.2 调用

```
func main() {
        var x int=100
        var y int=200
        swap(x,y)
        fmt.Println("x的值为：",x,"y的值为：",y)
}
func swap(numA,numB int){
        var tempVar int
        tempVar=numA
        numA=numB
        numB=tempVar
}
```

5.3 递归函数

```
/*输出斐波那契数列
 */
func main() {
        var array [10]int
        for i := 0; i < 10; i++ {
                array[i] = fibonacci(i)
        }
        fmt.Println(array)
}
```

```
func fibonacci(n int) (res int) {
        if n <= 1 {
                res = n
        }else{
                res = fibonacci(n-1) + fibonacci(n-2)
        }
        return
}
```

5.4 匿名函数

- 声明格式：

```
func ([params])([return_types]){
//函数体
}
```

5.4 匿名函数

- 调用示例：

```
func main() {
        func(str string) {
                fmt.Println(str)
        }("你好，三酷猫")
}
```

5.4 匿名函数

```
func main() {
//声明变量functionA，并将匿名函数赋值给它
	functionA:=func(str string) {
		fmt.Println(str)
	}
//通过functionA()调用匿名函数
	functionA("你好")
	functionA("三酷猫")
}
```

5.4 匿名函数

- 实现回调

```
func main(){
        //调用start()函数，传入匿名函数，输出当前时间
        start(func(){
                t:=time.Now()
                fmt.Println(t.Year(),t.Month(),t.Day(),t.Hour(),t.Minute(),t.Second())
        })
}
//命名函数start()，每隔一秒通过f()调用一次匿名函数
func start(f func()){
        for{
                //延迟1秒执行
                time.Sleep(1*time.Second)
                f()
        }
}
```

5.5 闭包

- 实现回调

```
func main() {
        accumulator := Accumulate(1)
        fmt.Println(accumulator())
        fmt.Println(accumulator())
        fmt.Println(accumulator())
        accumulator2 := Accumulate(10)
        fmt.Println(accumulator2())
        fmt.Println(accumulator2())
        fmt.Println(accumulator2())
accumulator3 := Accumulate(100)
        fmt.Println(accumulator3())
        fmt.Println(accumulator3())
        fmt.Println(accumulator3())
        //再次通过每个变量调用各自的函数
        fmt.Println(accumulator())
        fmt.Println(accumulator2())
        fmt.Println(accumulator3())
}
func accumulate(value int) func() int {
        return func() int {
                value++
                return value
        }
}
```

5.6 函数的延迟调用

- 关键字：defer

```
fmt.Println("defer行为开始")
defer fmt.Println("操作1")
defer fmt.Println("操作2")
defer fmt.Println("操作3")
fmt.Println("defer行为结束")
```

5.7 异常处理理

- 运行时宕机Panic

```
func main(){
        fmt.Println("程序开始执行")
        defer fmt.Println("发生宕机后要运行的逻辑")
        panic("发生宕机！")
        fmt.Println("程序停止执行")
}
```

5.7 异常处理理

- 宕机时恢复Recover

```
func main(){
        fmt.Println("程序开始执行")
        defer func() {
                err:=recover()
                if err=="USB设备被拔出"{
                        fmt.Println("USB设备被拔出，请重新插入")
                }
        }()
        panic("USB设备被拔出")
}
```