函数

实现一个简单的加减乘除运算

```
func eval(a, b int, op string) int {
    switch op {
    case "+":
        return a + b
    case "-":
        return a - b
    case "*":
        return a * b
    case "/":
        return a / b
    default:
        panic("unsupported operation:" + op)
    }
}
```

注意的地方是，返回值写在了参数列表的后面，虽然有一些不习惯，但是还是两个字：优雅！接下来不止是优雅，甚至是powerful的用法：

```
func div(a, b int) (q, r int) {
    return a/b,a%b
}
```

可以进行多返回值，那这也太棒了，再也不用一个数组存起来返回去了，强大！

甚至在调用函数的时候，可以用两个值进行承接，来我们看看

```
q, r := div(4, 3)
fmt.Println(q, r)
```

及其简洁的返回方式，但是我们也仍然发现了一个问题，在返回多变量的时候，我们给它甚至加入了变量名？？？？

当然这个用法可以这样：

```
func div(a, b int) (q, r int) {
    q=a/b
    r=a%b
    return
}
```

这个也是非常棒的操作

但是如果我们只想收到其中一个返回值呢？？？

```
a,_:=div(4,5)
```

只需要用_将原本的位置进行替代

接下来我们进行一个简单的异常处理

```
//函数体
func eval(a, b int, op string) (int, error) {
    switch op {
    case "+":
        return a + b, nil
    case "-":
        return a - b, nil
    case "*":
        return a * b, nil
    case "/":
        return a / b, nil
    default:
        return 0, fmt.Errorf("unsupported operation: %s", op)
    }
}
//main函数中的元素
if x, error := eval(3, 4, "+"); error != nil {
    fmt.Println(error)
} else {
    fmt.Println(x)
}
```

学到这里我已经够震惊了，接下来会更震惊一点的东西

```
func apply(op func(int, int) int, a, b int) int {
    p := reflect.ValueOf(op).Pointer()    //拿到函数的指针
    opName := runtime.FuncForPC(p).Name() //拿到函数的名字
    fmt.Printf("Calling function %s with args"+"(%d %d)", opName,
a, b)
        return op(a,b)
}
```

这里我们甚至将函数作为了参数进行操作，这里的用法其实是，将特定类型的函数作为参数传入进来，这样就可以更加方便调用这些函数，即利用一个名称进行调用。

可变参数列表

```
func sum(number ...int) int {
    s := 0
    for i := range number {
        s += number[i]
    }
    return s
}
```

这样操作也是十分的完美，这里还是非常强大