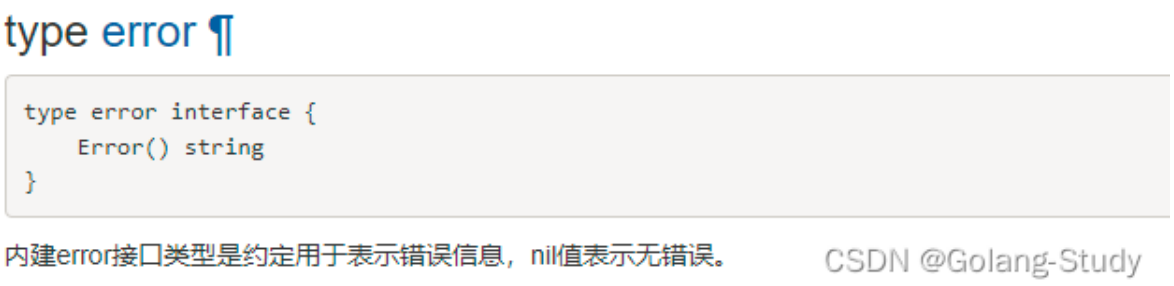
1、基本介绍

errors包实现了`创建错误值`的函数。



type error ¶

```
type error interface {
    Error() string
}
```

内建error接口类型是约定用于表示错误信息，nil值表示无错误。

2、创建错误函数

一个是errors包中的New函数，另一个是格式化错误信息，使用fmt包中的Errorf。

func New

```
func New(text string) error
```

使用字符串创建一个错误,请类比fmt包的Errorf方法，差不多可以认为是New(fmt.Sprintf(...))。

Example

```
err := errors.New("emit macho dwarf: elf header corrupted")
if err != nil {
    fmt.Print(err)
}
```

Output:

```
emit macho dwarf: elf header corrupted
```

Example (Errorf)

```
const name, id = "bimmler", 17
err := fmt.Errorf("user %q (id %d) not found", name, id)
if err != nil {
    fmt.Print(err)
}
```

当我们需要传入格式化的错误描述信息时，使用fmt.Errorf是个更好的选择。这个函数会返回一个错误对象

Output:

```
user "bimmler" (id 17) not found
```



```
// MyError is an error implementation that includes a time and message.
type MyError struct {
    When time.Time
    What string
}
func (e MyError) Error() string {
    return fmt.Sprintf( format: "%v: %v", e.When, e.What)
}
```

自定义错误结构体，实现Error方法即可

3、自定义错误

```
package main

import (
    "errors"
    "fmt"
)

// 函数去读取配置文件init.conf的信息
// 如果文件名传入不正确，就返回一个自定义错误
```

```go
func readConf(name string) (err error) {
    if name == "config.ini" {
        return nil
    } else {
        // 返回一个自定义错误
        return errors.New("name is not a valid configuration")
    }
}

func test() {

    defer func() {
        err := recover() // recover是内置函数，可以捕获到异常
        if err != nil {
            fmt.Println("err = ", err)
        }
    }()

    err := readConf("config111.ini")
    if err != nil {
        //如果读取文件错误，就输出错误信息，并终止程序（如果没有上面的defer处理，就会终止程序）
        panic(err)
    }
    fmt.Println("test继续执行")
}

func main() {

    //测试
    test()
    fmt.Println("MAIN...")
}
```

```
err =  name is not a valid configuration
MAIN...
```


将上面代码中的错误接口改成自定义的。

```go
package main

import (
    _"errors"
    "fmt"
    "time"
)

// MyError is an error implementation that includes a time and message.
type MyError struct {
    When time.Time
    What string
}
func (e MyError) Error() string {
    return fmt.Sprintf("%v: %v", e.When, e.What)
}
```

```go
// 函数去读取配置文件init.conf的信息
// 如果文件名传入不正确，就返回一个自定义错误
func readConf(name string) (err *MyError) {
    if name == "config.ini" {
        return nil
    } else {
        // 返回一个自定义错误
        return &MyError{
            time.Now(),
            "the file system has gone away",
        }
    }
}

func test() {

    defer func() {
        err := recover() // recover是内置函数，可以捕获到异常
        if err != nil {
            fmt.Println("err = ", err)
        }
    }()

    err := readConf("config111.ini")
    if err != nil {
        //如果读取文件错误，就输出错误信息，并终止程序（如果没有上面的defer处理，就会终止程序）
        panic(err)
    }
    fmt.Println("test继续执行")
}

func main() {

    //测试
    test()
    fmt.Println("MAIN...")
}
```

```
err =  2022-08-27 08:29:16.3814488 +0800 CST m=+0.003431601: the file system has gone away
MAIN...
```


4、总结

1、erors包中主要是定义了一个New函数，根据传入的参数信息生成一个error错误接口的对象(error错误类型在builtin包中已经预定义)。
2、在实际代码开发中，写一个业务函数，他有可能会出现问题，这是可以根据条件判断哪种情况是认为出现了错误的，此时就在这种情况下发挥一个error错误对象。
3、此时有两种错误类型，一种是预定以的error错误，另一种就是自定义的错误。如果使用预定义的error，那么有两种方式，一种是直接用errors包中的New返回一个error对象，另一种是fmt包中的Errorf函数。
4、此时，有一个函数或者方法需要调用上面的业务函数，此时根据返回值error就可以判断出是否返回了错误，如果存在错误，那么就在这个判断中使用panic(err)函数，把返回的错误对象传给panic函数。
5、在当前函数或者方法中，因为调用了panic函数，当前代码会强制中断。如果想让程序不会因为这个

错误而终止，当前函数中就需要使用defer延迟机制，让延迟的函数去调用recover函数，该函数会返回一个error对象，这个就是panic传过来的参数。如果返回值不是nil，那么就要处理这个错误即可。
6、因为当前函数或者方法已经因为发生错误，最后去执行了defer中的函数，那么当前的这个函数或者方法就已经终止了，后面的代码不会执行，但是主程序还是可以往下继续执行。

其他：
在平常使用的go中的函数或者方法，很多都会有返回值是error对象的情况，为了避免程序终止，可以在调用该函数或方法的函数中使用defer延迟机制，执行函数，在里面通过recover函数监测是否发生了错误，进行错误处理。
或者说，可以在获取了error返回值之后，通过判断是否为nil为空，直接处理这个处理，让当前函数继续执行或者直接return。