Go语言 标准库 Errors包

errors包实现了操作错误的函数。语言使用 `error` 类型来返回函数执行过程中遇到的错误，如果返回的 error 值为 `nil`，则表示未遇到错误，否则 error 会返回一个字符串，用于说明遇到了什么错误。

error的结构

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

你可以用任何类型去实现它（只要添加一个 Error() 方法即可），也就是说，error 可以是任何类型，这意味着，函数返回的 error 值实际可以包含任意信息，不一定是字符串。

error不一定表示一个错误，它可以表示任何信息，比如io包中就用error类型的io.EOF表示数据读取结束，而不是遇到了什么错误。

errors包实现了一个最简单的error类型，只包含一个字符串，它可以记录大多数情况下遇到的错误信息。errors包的用法也很简单，只有一个New函数，用于生成一个最简单的error对象：

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

简单使用

```
package main

import (
    "errors"
    "fmt"
)

func check(s string) (string, error) {
    if s == "" {
        err := errors.New("字符串不能为空")
        return "", err
    } else {
        return s, nil
    }
}

func main() {
    s, err := check("")
    if err != nil {
        fmt.Printf("err: %v\n", err.Error())
    } else {
        fmt.Printf("s: %v\n", s)
    }
}
#结果
err: 字符串不能为空
```

自定义错误

Go允许函数具有多返回值，但通常你不会想写太多的返回值在函数定义上，而标准库内置的errorString类型由于只能表达字符串错误信息显然受限。所以，可以通过实现error接口的方式，来扩展错误返回。

```
package main

import (
    "fmt"
)

// 自定义error类型
type MyError struct {
    Msg  string // 错误文本信息
    Code int64  // 错误码
}

func (e *MyError) Error() string {
    // 当然，你也可以自定义返回的string，比如
    return fmt.Sprintf("code %d, msg %s", e.Code, e.Msg)
}

// 实现了error接口
func DoSomething() error {
    return &MyError{"my error", 1}
}

// 业务应用
func DoBusiness() {
    err := DoSomething()
    e, ok := err.(*MyError)
    if ok {
        fmt.Printf("code %d, msg %s\n", e.Code, e.Msg)
    }
}

func main() {
    DoBusiness()
}
```