标识符

标识符用来命名变量、类型等程序实体（包括常量、变量、函数、结构体、数组、切片等等）。一个**标识符实际上就是一个或是多个字母(A~Z和a~z)数字(0~9)、下划线_组成的序列**，但是**第一个字符必须是字母或下划线而不能是数字。标识符区分大小写。**文件名不包含空格或其他特殊字符。以下为有效标识符：

```
baozi   arr    arr1   arr_1    _arr2
a       Myarr  myArr  my23arr
```

以下为无效标识符：

```
1baozi   以数字开头
case     系统关键字
a+b      运算符是不允许的
```

`_` 本身就是一个特殊的标识符，被称为空白标识符。它可以像其他标识符那样用于变量的声明或赋值（任何类型都可以赋值给它），但任何赋给这个标识符的值都将被抛弃，因此这些值不能在后续的代码中使用，也不可以使用这个标识符作为变量对其它变量进行赋值或运算。

字符串连接

Go 语言的字符串连接可以通过 **+** 实现：

```
package main
import "fmt"
func main() {
    fmt.Println("Hello" + "Baozi")
}

#实际输出结果：HelloBaozi
```

关键字

下面列举了 Go 代码中会使用到的 25 个关键字或保留字：

break	**default**	**func**	**interface**	**select**
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

除了以上介绍的这些关键字，Go 语言还有 36 个预定义标识符：

append	bool	byte	cap	close	complex	complex64	complex128	uint16
copy	false	float32	float64	imag	int	int8	int16	uint32
int32	int64	iota	len	make	new	nil	panic	uint64
print	println	real	recover	string	true	uint	uint8	uintptr

程序一般由关键字、常量、变量、运算符、类型和函数组成。

程序中可能会使用到这些分隔符：括号 ()，中括号 [] 和大括号 {}。

程序中可能会使用到这些标点符号：.、,、;、: 和 …。

程序的代码通过语句来实现结构化。每个语句不需要像 C 家族中的其它语言一样以分号 `;` 结尾，因为这些工作都将由 Go 编译器自动完成。

如果你打算将多个语句写在同一行，它们则必须使用 `;` 人为区分，但在实际开发中我们并不鼓励这种做法。

Go中的命名规范

- ## 命名规范

- ### Go是一门区分大小写的语言。

命名规则涉及变量、常量、全局函数、结构、接口、方法等的命名。 Go语言从语法层面进行了以下限定：任何需要对外暴露的名字必须以大写字母开头，不需要对外暴露的则应该以小写字母开头。

1. 当命名（包括常量、变量、类型、函数名、结构字段等等）以一个大写字母开头，如：Analysize，那么使用这种形式的标识符的对象就**可以被外部包的代码所使用**（客户端程序需要先导入这个包），这被称为导出（像面向对象语言中的 public）；
2. **命名如果以小写字母开头，则对包外是不可见的，但是他们在整个包的内部是可见并且可用的**（像面向对象语言中的 private ）

- ### 包名称

保持package的名字和目录保持一致，尽量采取有意义的包名，简短，有意义，尽量和标准库不要冲突。包名应该为**小写**单词，不要使用下划线或者混合大小写。

```
package domain
package main
```

- ### 文件命名

尽量采取有意义的文件名，简短，有意义，应该为**小写**单词，使用**下划线**分隔各个单词。

```
approve_service.go
```

- **结构体命名**

1. 采用**驼峰命名法**，首字母根据访问控制大写或者小写
2. struct 申明和初始化格式采用多行，例如下面：

```
type MainConfig struct {
    Port string `json:"port"`
    Address string `json:"address"`
}
config := MainConfig{"1234", "123.221.134"}
```

- **接口命名**

1. 命名规则基本和上面的结构体类型
2. 单个函数的结构名以 “er” 作为后缀，例如 Reader , Writer 。

```
type Reader interface {
        Read(p []byte) (n int, err error)
}
```

- **变量命名**

和结构体类似，变量名称一般遵循驼峰法，首字母根据访问控制原则大写或者小写，但遇到特有名词时，需要遵循以下规则：

1. 如果变量为私有，且特有名词为首个单词，则使用小写，如 appService
2. 若变量类型为 bool 类型，则名称应以 Has, Is, Can 或 Allow 开头

```
var isExist bool
var hasConflict bool
var canManage bool
var allowGitHook bool
```

- **常量命名**

常量均需使用全部大写字母组成，并使用下划线分词

```
const APP_URL = "https://www.baidu.com"
```

如果是枚举类型的常量，需要先创建相应类型：

```
type Scheme string

const (
    HTTP  Scheme = "http"
    HTTPS Scheme = "https"
)
```

错误处理

- 错误处理的原则就是不能丢弃任何有返回err的调用，不要使用 _ 丢弃，必须全部处理。接收到错误，要么返回err，或者使用log记录下来

- 尽早return：一旦有错误发生，马上返回
- 尽量不要使用panic，除非你知道你在做什么
- 错误描述如果是英文必须为小写，不需要标点结尾
- 采用独立的错误流进行处理

```
// 错误写法
if err != nil {
    // error handling
} else {
    // normal code
}

// 正确写法
if err != nil {
    // error handling
    return // or continue, etc.
}
// normal code
```

单元测试

单元测试文件名命名规范为 example_test.go 测试用例的函数名称必须以 Test 开头，例如：TestExample 每个重要的函数都要首先编写测试用例，测试用例和正规代码一起提交方便进行回归测试。