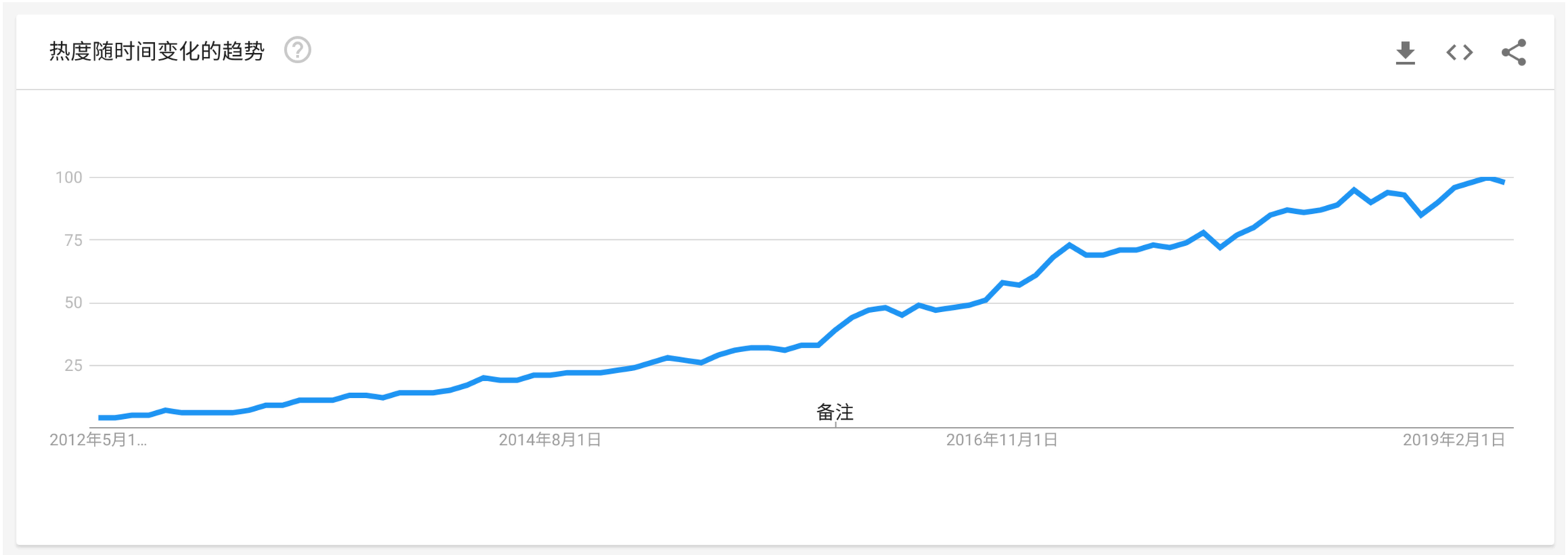
Golang 实战

优点知识 - 阳明

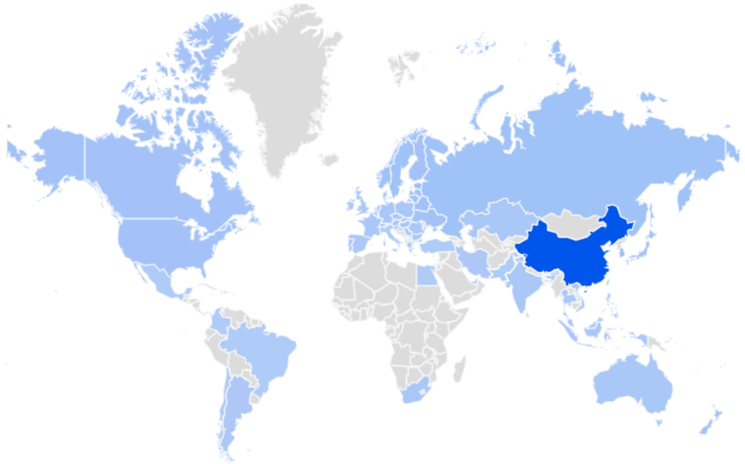
https://youdianzhishi.com


热度随时间变化的趋势
100
75
50
25
备注
2012年5月1...
2014年8月1日
2016年11月1日
2019年2月1日

Golang 趋势

按区域显示的搜索热度

区域



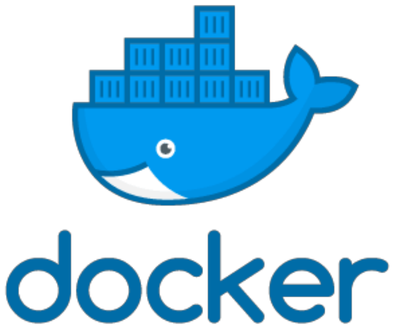
1	中国	100
2	圣赫勒拿	16
3	新加坡	16
4	韩国	12
5	香港	11

包括搜索量较低的区域

当前显示的是第 1-5 个地区（共 61 个）

Golang 中国

Golang 项目



课程设计

- 基本语法
- 面向接口编程
- 函数式编程
- 并发编程（goroutine+channel）
- 文本处理（工具）
- Nginx 日志处理（协程）
- Go Web（web）

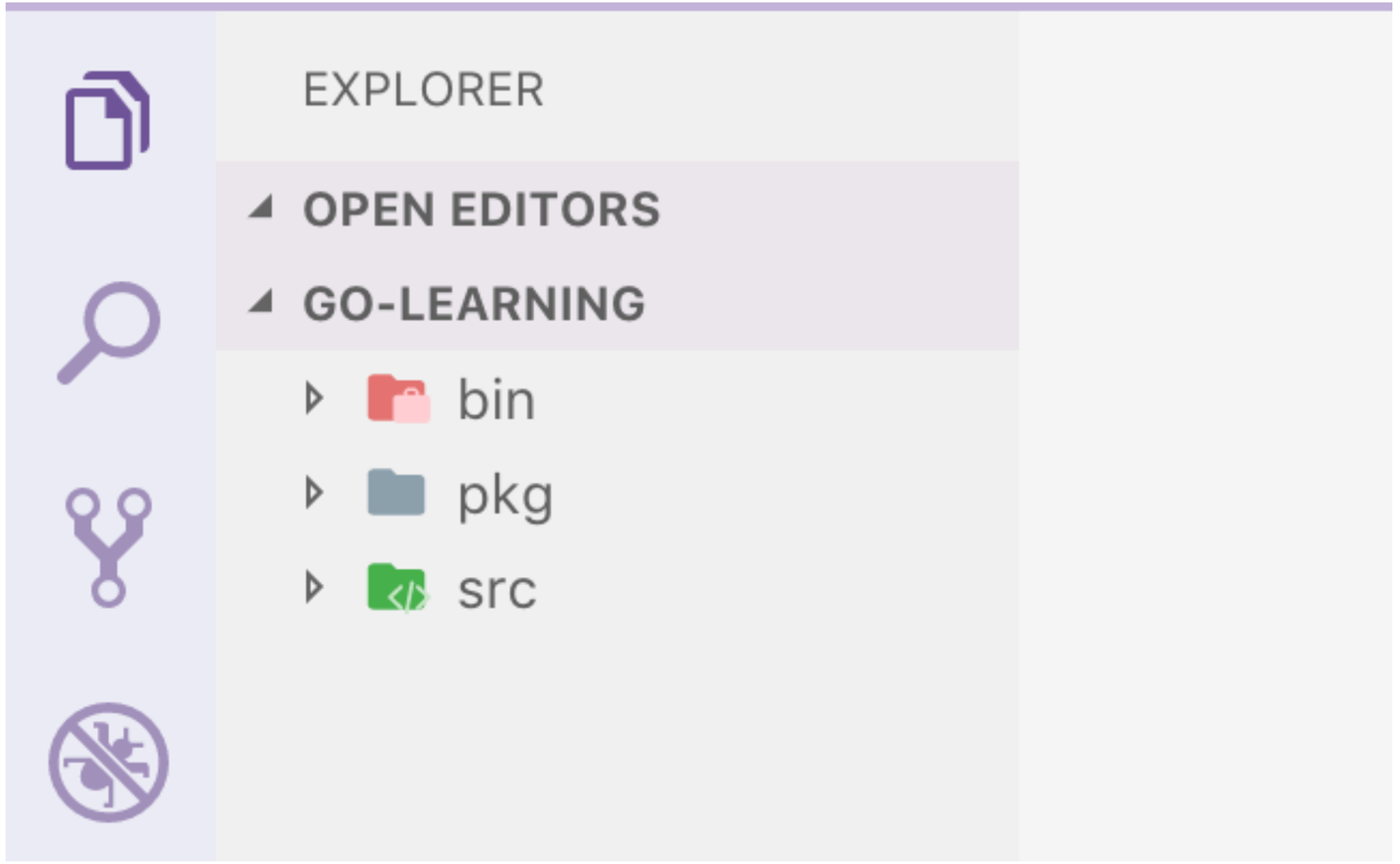
Golang 安装&开发工具

- 安装：https://golang.google.cn/dl/
- 开发工具：vim、emacs、sublime text、atom、vscode(推荐1)、GoLand(推荐2)

Golang 开发环境

- GOROOT：go 的安装目录，设置这个环境变量自定义 go 路径
- GOPATH：go 的工作目录（项目目录），编译或运行时从这个环境变量中去查找包、依赖

Go 项目基本结构



Go语言基本形式

```go
package main ❶

import "fmt" ❷

func main() {
    fmt.Println("Hello World") ❸
}
```

❶ 使用 package 关键字定义包

❷ 导入打印需要用到的系统包：fmt

❸ 调用系统包 fmt 中的 Println 方法打印字符串

构建并运行

- go build：编译指定的源文件或代码包以及依赖包
- go install：安装自身包和依赖包
- go run：编译并运行 go 程序
- vscode 插件：Code Runner
- GoLand IDE

Go tools 配置

```
{
    "go.gopath": "${workspaceFolder}",
    "go.inferGopath": true,
    "go.autocompleteUnimportedPackages": true,
    "go.gocodePackageLookupMode": "go",
    "go.gotoSymbol.includeImports": true,
    "go.useCodeSnippetsOnFunctionSuggest": true,
    "go.useCodeSnippetsOnFunctionSuggestWithoutType": true,
    "go.docsTool": "gogetdoc"
}
```

1. "cmd+shift+p": “go:install/update tools” 安装插件
2. "go.toolsGopath": “指定tools包的路径，不指定默认在 GOPATH”

变量

```
var a int、var a, b int = 1, 2
var a, b, c = 1, "go", true
a, b, c := 1, "go", true ❶
var (
    a = 1
    b = "go"
    c = true
)
```

❶ 只能在函数内使用，不能用在函数外

系统变量类型

- bool
- string
- (u)int、(u)int8、(u)int16、(u)int32、(u)int64、uintptr
- byte(uint8)、rune(int32 unicode)
- float32、float64、complex64、complex128

1. Go语言不允许隐式类型转换，只能强制转换
2. Go语言中不支持指针运算

常量

```
const s string = "Hello"
const a, b = 3, 4  ❶
const (
    s1  = "golang"
    c   = 5
    MAX = 10        ❷
)
```

.....

❶ 不指定类型的常量，则它的类型是不确定的,可以做各种类型使用

❷ Go语言中定义常量一般不用大写，大写表示public，可导出的，有特殊含义

枚举

```
const (
    Monday = 1 + iota  ❶
    Tuesday
    Wednesday
    Thursday
    Friday
    Staurday
    Sunday
)
```

❶ 枚举是一种特殊的常量，可以通过iota快速设置连续的值

类型定义与别名

```
type MyInt1 int                ❶
type MyInt2 = int              ❷

var i int = 1
var i1 MyInt1 = MyInt1(i)      ❸
var i2 MyInt2 = i
```

❶ 类型定义：基于int创建的一个新类型，主要提高代码可读性

❷ 类型别名：基于int创建的一个别名，和原类型完全一样，主要用于包兼容

❸ 类型定义是一个新的类型了，所以类型转换的时候必须强制类型转换

算术运算符

下表列出了所有Go语言的算术运算符。假定 A 值为 10，B 值为 20。

运算符	描述	实例
+	相加	A + B 输出结果 30
-	相减	A - B 输出结果 -10
*	相乘	A * B 输出结果 200
/	相除	B / A 输出结果 2
%	求余	B % A 输出结果 0
++	自增	A++ 输出结果 11 [1]
--	自减	A-- 输出结果 9 [1]

[1] Go语言中没有前置的++和--，++a、--a是错误的

关系运算符

下表列出了所有Go语言的关系运算符。假定 A 值为 10，B 值为 20。

运算符	描述	实例
==	检查两个值是否相等，如果相等返回 True 否则返回 False。	(A == B) 为 False
!=	检查两个值是否不相等，如果不相等返回 True 否则返回 False。	(A != B) 为 True
>	检查左边值是否大于右边值，如果是返回 True 否则返回 False。	(A > B) 为 False
<	检查左边值是否小于右边值，如果是返回 True 否则返回 False。	(A < B) 为 True
>=	检查左边值是否大于等于右边值，如果是返回 True 否则返回 False。	(A >= B) 为 False
<=	检查左边值是否小于等于右边值，如果是返回 True 否则返回 False。	(A <= B) 为 True

逻辑运算符

下表列出了所有Go语言的逻辑运算符。假定 A 值为 True，B 值为 False。

运算符	描述	实例
&&	逻辑 AND 运算符。 如果两边的操作数都是 True，则条件 True，否则为 False。	(A && B) 为 False
||	逻辑 OR 运算符。 如果两边的操作数有一个 True，则条件 True，否则为 False。	(A || B) 为 True
!	逻辑 NOT 运算符。 如果条件为 True，则逻辑 NOT 条件 False，否则为 True。	!(A && B) 为 True

位运算符

下表列出了位运算符 &, |, 和 ^ 的计算：

p	q	p & q	p	q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Go 语言支持的位运算符如下表所示。假定 A 为60，B 为13：

运算符	描述	实例
&	按位与运算符"&"是双目运算符。 其功能是参与运算的两数各对应的二进位相与。	(A & B) 结果为 12, 二进制为 0000 1100
|	按位或运算符"|"是双目运算符。 其功能是参与运算的两数各对应的二进位相或	(A | B) 结果为 61, 二进制为 0011 1101
^	按位异或运算符"^"是双目运算符。 其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为1。	(A ^ B) 结果为 49, 二进制为 0011 0001
<<	左移运算符"<<"是双目运算符。左移n位就是乘以2的n次方。 其功能把"<<"左边的运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。	A << 2 结果为 240 ， 二进制为 1111 0000
>>	右移运算符">>"是双目运算符。右移n位就是除以2的n次方。 其功能是把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数。	A >> 2 结果为 15 ， 二进制为 0000 1111

赋值运算符

运算符	描述	实例
=	简单的赋值运算符，将一个表达式的值赋给一个左值	C = A + B 将 A + B 表达式结果赋值给 C
+=	相加后再赋值	C += A 等于 C = C + A
-=	相减后再赋值	C -= A 等于 C = C - A
*=	相乘后再赋值	C *= A 等于 C = C * A
/=	相除后再赋值	C /= A 等于 C = C / A
%=	求余后再赋值	C %= A 等于 C = C % A
<<=	左移后赋值	C <<= 2 等于 C = C << 2
>>=	右移后赋值	C >>= 2 等于 C = C >> 2
&=	按位与后赋值	C &= 2 等于 C = C & 2
^=	按位异或后赋值	C ^= 2 等于 C = C ^ 2
|=	按位或后赋值	C |= 2 等于 C = C | 2

条件语句

```
if 布尔表达式 {
    /* 在布尔表达式为 true 时执行 */
}
```

```
if 布尔表达式 {
    /* 在布尔表达式为 true 时执行 */
} else if 另外一个布尔表达式 {
    /* 在布尔表达式为 true 时执行 */
} else {
    /* 在布尔表达式为 false 时执行 */
}
```

条件语句

```
switch {
case true:                          ❶
        fmt.Println("1、case 条件语句为 false”)
        fallthrough                 ❷
case false:
        fmt.Println("2、case 条件语句为 false")
case true:
        fmt.Println("3、case 条件语句为 true")
case true:
        fmt.Println("4、case 条件语句为 true")
default:
        fmt.Println(“5、默认 case”)
}
```

❶ 直到找到匹配项，匹配项后面也不需要加 break，相当于默认就有 break

❷ 默认情况下 case 匹配成功后就不会执行其他 case，如果我们需要执行后面的 case，可以使用 fallthrough，fallthrough 不会判断下一条 case 的表达式结果是否为 true。