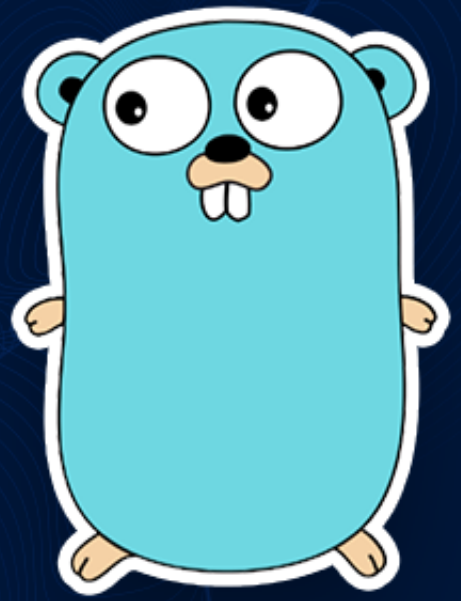
从 Python 到 Go:
动态与静态语言之间如何选型



张冶青 Marvin　2020-12-17

目录

Python: 程序员的瑞士军刀

概览

- 创始者: Guido van Rossum
- 发行时间: 1991年
- 语言特点: 优雅、明确、简单
- 主要应用领域（不限于）:
  - 人工智能/机器学习
  - 科学计算
  - 数据分析
  - 自动化运维
  - 后端
  - 爬虫
- 相关名言:
  - “人生苦短，我用 Python”
  - “动态一时爽，重构火葬场”
  - 《Python 之禅》

```
~ » python -c "import this"
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

```
Python之禅 by Tim Peters

优美胜于丑陋（Python 以编写优美的代码为目标）
明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）
简洁胜于复杂（优美的代码应当是简洁的，不要有复杂的内部实现）
复杂胜于凌乱（如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁）
扁平胜于嵌套（优美的代码应当是扁平的，不能有太多的嵌套）
间隔胜于紧凑（优美的代码有适当的间隔，不要奢望一行代码解决问题）
可读性很重要（优美的代码是可读的）
即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）

不要包容所有错误，除非你确定需要这样做（精准地捕获异常，不写 except:pass 风格的代码）

当存在多种可能，不要尝试去猜测
而是尽量找一种，最好是唯一一种明显的解决方案（如果不确定，就用穷举法）
虽然这并不容易，因为你不是 Python 之父（这里的 Dutch 是指 Guido ）

做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）

如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）

命名空间是一种绝妙的理念，我们应当多加利用（倡导与号召）
```

Python: 程序员的瑞士军刀

主要应用领域

人工智能/机器学习

- Tensorflow
- PyTorch
- Scikit-Learn
- Nltk

数据分析

- Pandas
- Statsmodels
- Matplotlib
- Seaborn

后端

- Django
- Flask
- Tornado
- FastApi

科学计算

- Numpy
- Scipy
- Sympy
- Jupyter

自动化运维

- Ansible
- Fabric
- SaltStack
- Supervisor

爬虫

- Scrapy
- PySpider
- Selenium
- Pyppeteer

Python: 程序员的瑞士军刀

Talk is cheap, show me the code

```python
# 引用依赖
import random

# 生成列表
items = [random.randint(0, 10) for i in range(10)]
print(items)
# >>> [4, 7, 1, 5, 1, 9, 0, 9, 2, 8]

# 列表排序
items_sorted = sorted(items) # 顺序
print(items_sorted)
# >>> [0, 1, 1, 2, 4, 5, 7, 8, 9, 9]
items_sorted_reverse = sorted(items, reverse=True) # 倒序
print(items_sorted_reverse)
# >>> [9, 9, 8, 7, 5, 4, 2, 1, 1, 0]

# 筛选列表 & lambda 表达式
items_filtered = filter(lambda d: d**2 < 9, items)
print(items_filtered)
# >>> [1, 1, 0, 2]

# 列表选择
items_first5 = items[:5] # 选择前5个元素
print(items_first5)
# >>> [4, 7, 1, 5, 1]
item_last = items[-1] # 选择最后一个元素
print(item_last)
# >>> 8
items_mapped = map(lambda d: d**2, items) # 每个元素平方
print(items_mapped)
# >>> [16, 49, 1, 25, 1, 81, 0, 81, 4, 64]
```

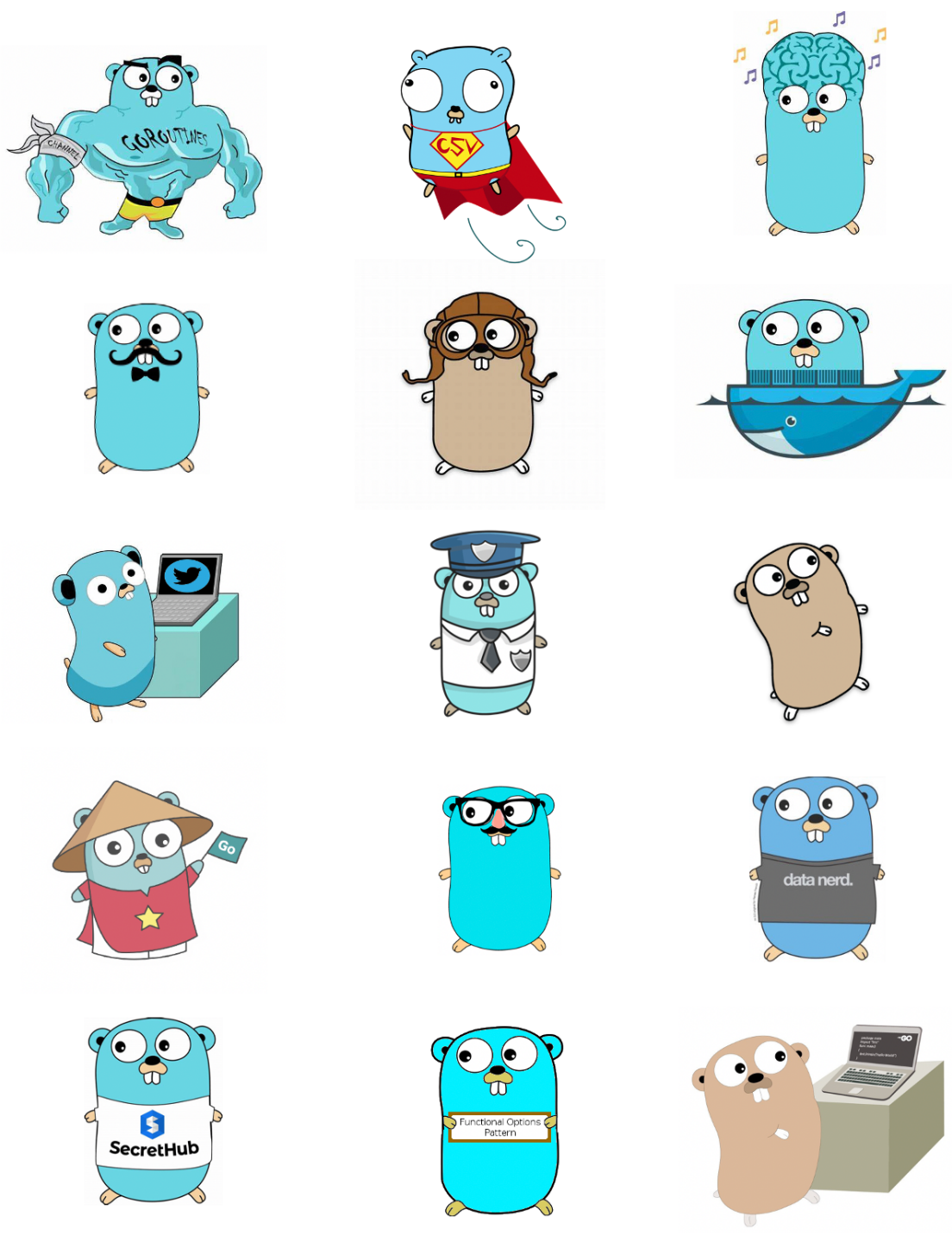
```python
# 定义函数
def my_func(arg1, arg2):
    print(arg1)
    print(arg2)

# 定义类
class MyClass(object):
    # 构造函数
    def __init__(self, arg1, arg2):
        self.value1 = arg1
        self.value2 = arg2

    # 定义公共方法
    def public_method():
        my_func(self.value1, self.value2)

    # 定义私有方法
    def _private_method():
        my_func(self.value1, self.value2)

# 继承类
class MyNewClass(MyClass):
    # 新公共方法
    def new_public_method():
        pass

# 生成实例
my_instance = MyClass(1, 2)
my_new_instance = MyNewClass(1, 2)

# 调用类方法
my_instance.public_method()
```

```python
# 字典
dict1 = {'a': 1, 'b': 2}

# 集合
set1 = set([1, 1, 2, 2])

# for 循环
items = range(10)
for i in items:
    print(i)

# while 循环
i = 0
while i < 10:
    print(i)
    i += 1

# if else
i = 1
if i % 2 == 0:
    print('even')
else:
    print('odd')

# try except
i = 0
try:
    i = 10 / i
except Exception as ex:
    print(ex)
```

Go: 高效精致的工程套装

概览

- 创始者: Google
- 发行时间: 2007年
- 语言特点: 简洁、快速、天然支持并发、编译迅速
- 主要应用领域（不限于）:
  - 容器
  - 后端
  - 分布式系统
  - 数据库
  - 爬虫
  - 机器学习
- 相关名言:
  - “不要恐慌（panic）”
  - “泛型去哪儿”
  - “err != nil”



Go: 高效精致的工程套装

主要应用领域

容器
- Docker
- Kubernetes
- Containerd
- Podman

分布式系统
- Ectd
- Consul
- SeaweedFS
- Micro

爬虫
- Colly
- Crawlab
- Ferret
- GoQuery

后端
- Gin
- Beego
- Echo
- Iris

数据库
- TiDB
- InfluxDB
- Prometheus
- LevelDB

机器学习
- GoLearn
- GoML
- Gorgonia
- Goro

Go: 高效精致的工程套装

Talk is cheap, show me the code

```go
package main

import (
    "fmt"
    "math"
    "math/rand"
    "sort"
)

func main() {
    // 生成列表
    var items []int
    for i := 0; i < 10; i++ {
        items = append(items, rand.Intn(10))
    }
    fmt.Println(items) // [1 7 7 9 1 8 5 0 6 0]

    // 列表排序
    sort.Ints(items)   // 顺序
    fmt.Println(items) // [0 0 1 1 5 6 7 7 8 9]
    // 倒序
    var itemsReverse []int
    for i := len(items) - 1; i >= 0; i-- {
        itemsReverse = append(itemsReverse, items[i])
    }
    fmt.Println(itemsReverse) // [9 8 7 7 6 5 1 1 0 0]

    // 筛选列表
    var itemsFiltered []int
    for _, item := range items {
        if math.Pow(float64(item), 2) < 9 {
            itemsFiltered = append(itemsFiltered, item)
        }
    }
    fmt.Println(itemsFiltered) // [0 0 1 1]

    // 列表选择
    itemsFirst5 := items[:5]         // 选择前5个元素
    fmt.Println(itemsFirst5)         // [0 0 1 1 5]
    itemLast := items[len(items)-1] // 选择最后一个元素
    fmt.Println(itemLast)            // 8
    // 每个元素的平方
    var itemsMapped []int
    for _, item := range items {
        itemsMapped = append(itemsMapped, int(math.Pow(float64(item), 2)))
    }
    fmt.Println(itemsMapped) // [0 0 1 1 25 36 49 49 64 81]
}
```

```go
package main

import "fmt"

// 定义公有函数
func PublicFunc(arg1 int, arg2 int) (res int) {
    res = arg1 + arg2
    return
}

// 定义私有函数
func privateFunc(arg1 int, arg2 int) (res int) {
    res = arg1 + arg2
    return
}

// 定义接口
type MyInterface interface {
    // 声明公有方法
    MyMethod() error
}

// 定义类
type MyClass struct {
    // 定义变量
    Variable int
}

// 实现类方法
func (c *MyClass) MyMethod() error {
    fmt.Println("call method")
    return nil
}

func main() {
    // 构造实例
    myInstance := MyClass{
        Variable: 0,
    }
    // 调用方法
    _ = myInstance.MyMethod()
}
```

```go
package main

import (
    "fmt"
    "github.com/pkg/errors"
)

func asyncFunc(ch chan string) {
    fmt.Println("call asyncFunc")

    // 向 channel 传值
    ch <- "finish"
}

func syncFunc(arg string) (err error) {
    fmt.Println("call syncFunc")

    if arg == "" {
        // 返回错误
        return errors.New("arg is invalid")
    }

    // 返回 nil，即没有错误，执行成功
    return nil
}

func main() {
    // 构造阻塞 channel
    ch := make(chan string)

    // 起 goroutine 异步调用函数，不阻塞
    go asyncFunc(ch)

    // 接收 channel 传值，阻塞
    res := <- ch
    fmt.Println(res) // 输出: finish

    // 用函数返回的错误值是否 (err != nil) 为空来判断是否成功执行
    if err := syncFunc("argument"); err != nil {
        // 如果错误，则执行该代码，相当于 except
        fmt.Println("error: " + err.Error())
        // 不会执行
    }
    if err := syncFunc(""); err != nil {
        fmt.Println("error: " + err.Error())
        // 输出: error: arg is invalid
    }
}
```

Python 和 Go 如何选型

属性	Python	Go
语言类别	动态语言	静态语言
需要编译	否	是
语法简洁	高	中
并发支持	中	高
OOP支持	高	中
运行效率	低	高
开发效率	高	中
可维护性	中	高
代码可读性	中	中
生态丰富度	高	中

Python 适合的场景

- 开发周期短
- 项目复杂度低
- 灵活度要求高
- 产品原型开发
- 性能要求不高

Go 适合的场景

- 开发周期长
- 项目较为复杂
- 可维护性要求高
- 并发要求高
- 运行效率要求高

学习资料推荐: Python

基础

- https://github.com/jackfrued/Python-Core-50-Courses (中文)
- https://github.com/learnbyexample/Python_Basics (英文)

进阶

- 《Python高性能编程》 - Micha Gorelick & Ian Ozsvald
- 《Python核心编程》 - Wesley Chun
- 《利用Python进行数据分析》 - Wes McKinney
- 《Python自然语言处理》 - Steven Bird, Ewan Klein & Edward Loper
- 《Python 3网络爬虫开发实战》 - 崔庆才
- 《Python 3网络爬虫宝典》 - 韦世东
- 极客时间《Python核心技术与实战》 -景霄

学习资料推荐: Go

基础

- 掘金小册《基于 Go 语言构建企业级的 RESTful API 服务》 - 雷克斯
- 《Go语言核心编程》 - 李文塔

进阶

- 《Go语言实战》 - William Kennedy, Brian Ketelsen & Erik St. Martin
- 《Go语言高级编程》 - 柴树杉 &曹春晖

Q & A