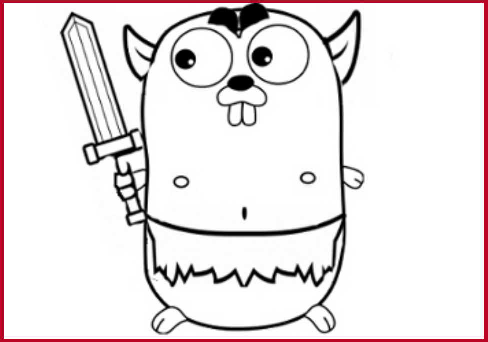


GO 性能优化

By @miraclesu

舜飞科技

网络运营全流程解决方案供应商

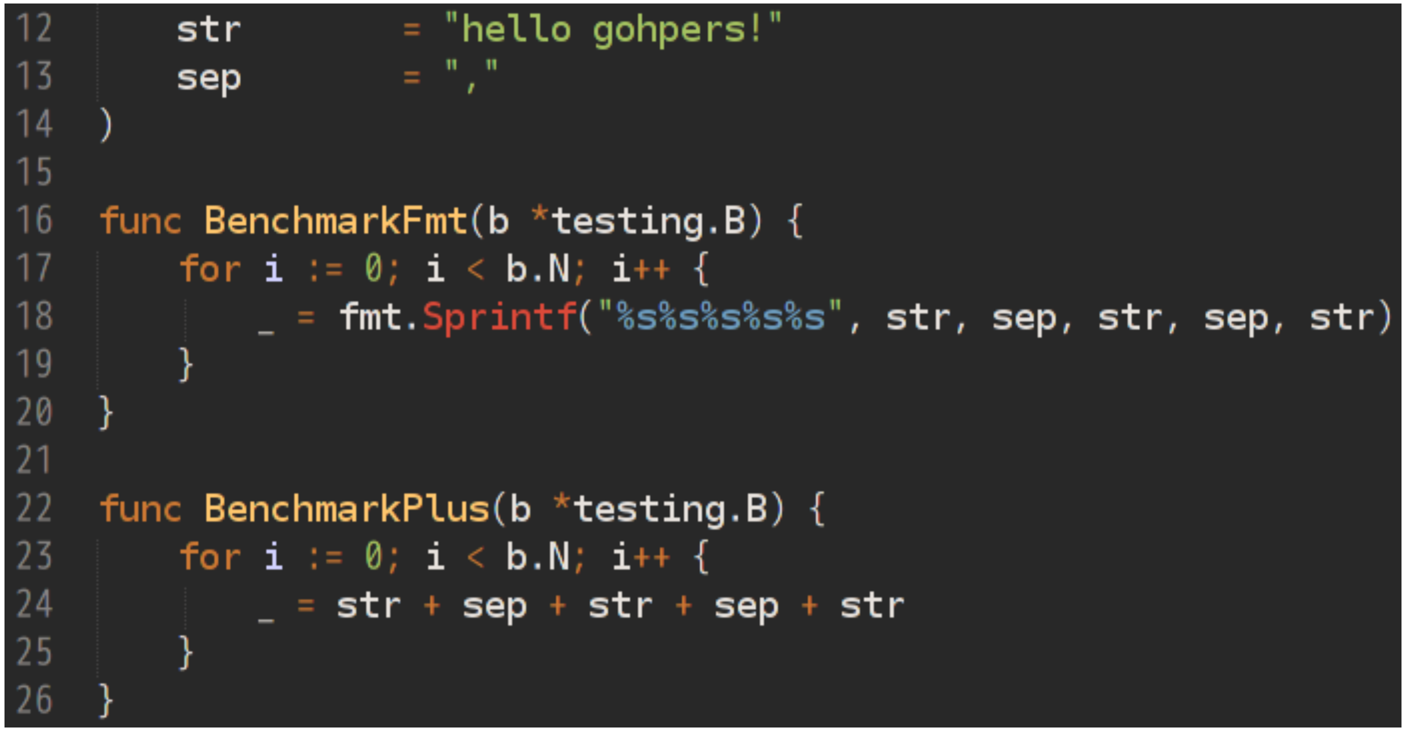
概要

- string & profiling
- slice & array
- slice & map 初始化
- 并发
- 缓存

string & profiling

string 连接1

fmt VS "+"

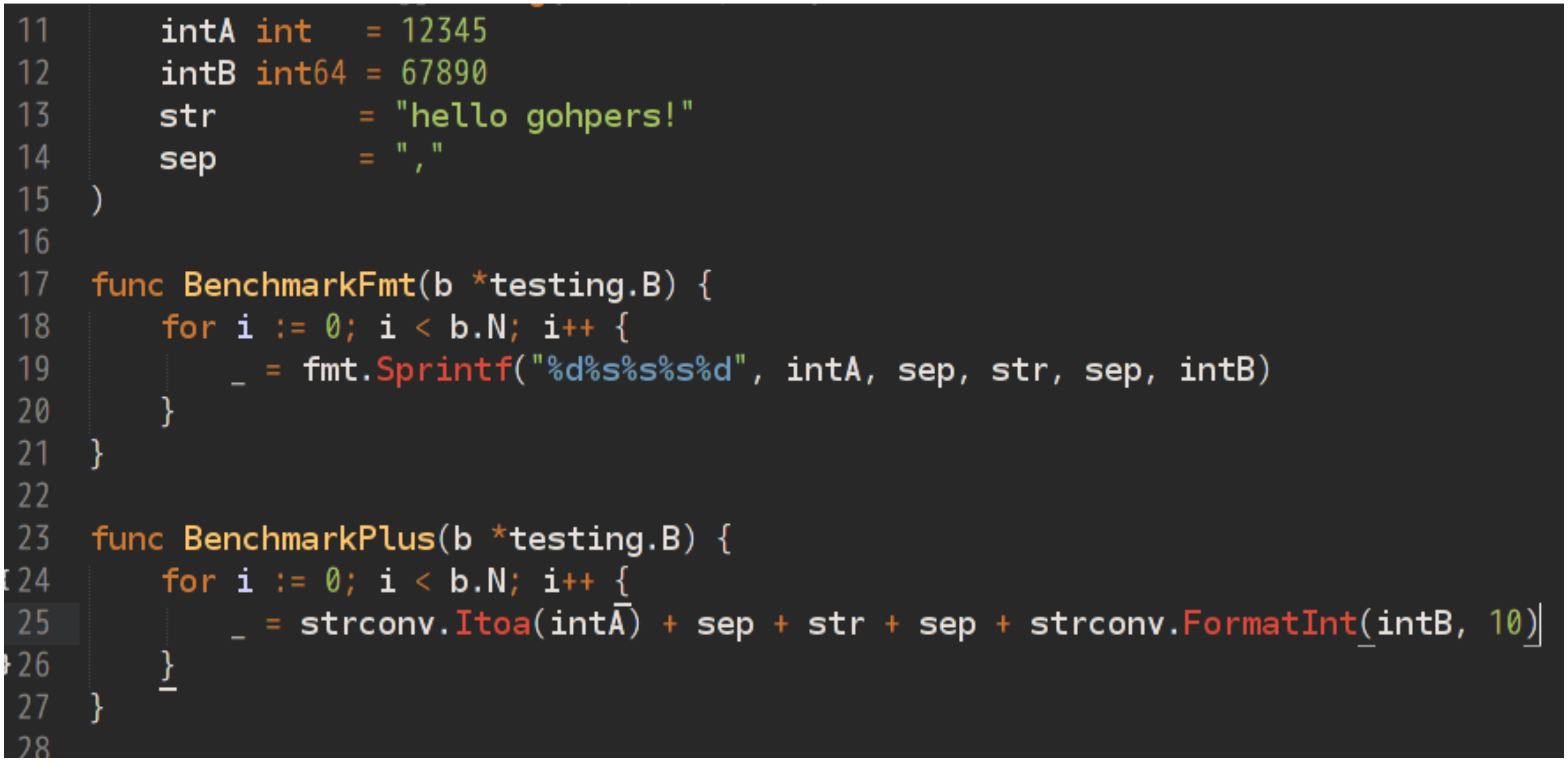


```go
    str         = "hello gohpers!"
    sep         = ","
)

func BenchmarkFmt(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = fmt.Sprintf("%s%s%s%s%s", str, sep, str, sep, str)
    }
}

func BenchmarkPlus(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = str + sep + str + sep + str
    }
}
```

```
BenchmarkFmt     1000000     1617 ns/op
BenchmarkPlus    5000000      393 ns/op
```

string 连接2

fmt VS "+"

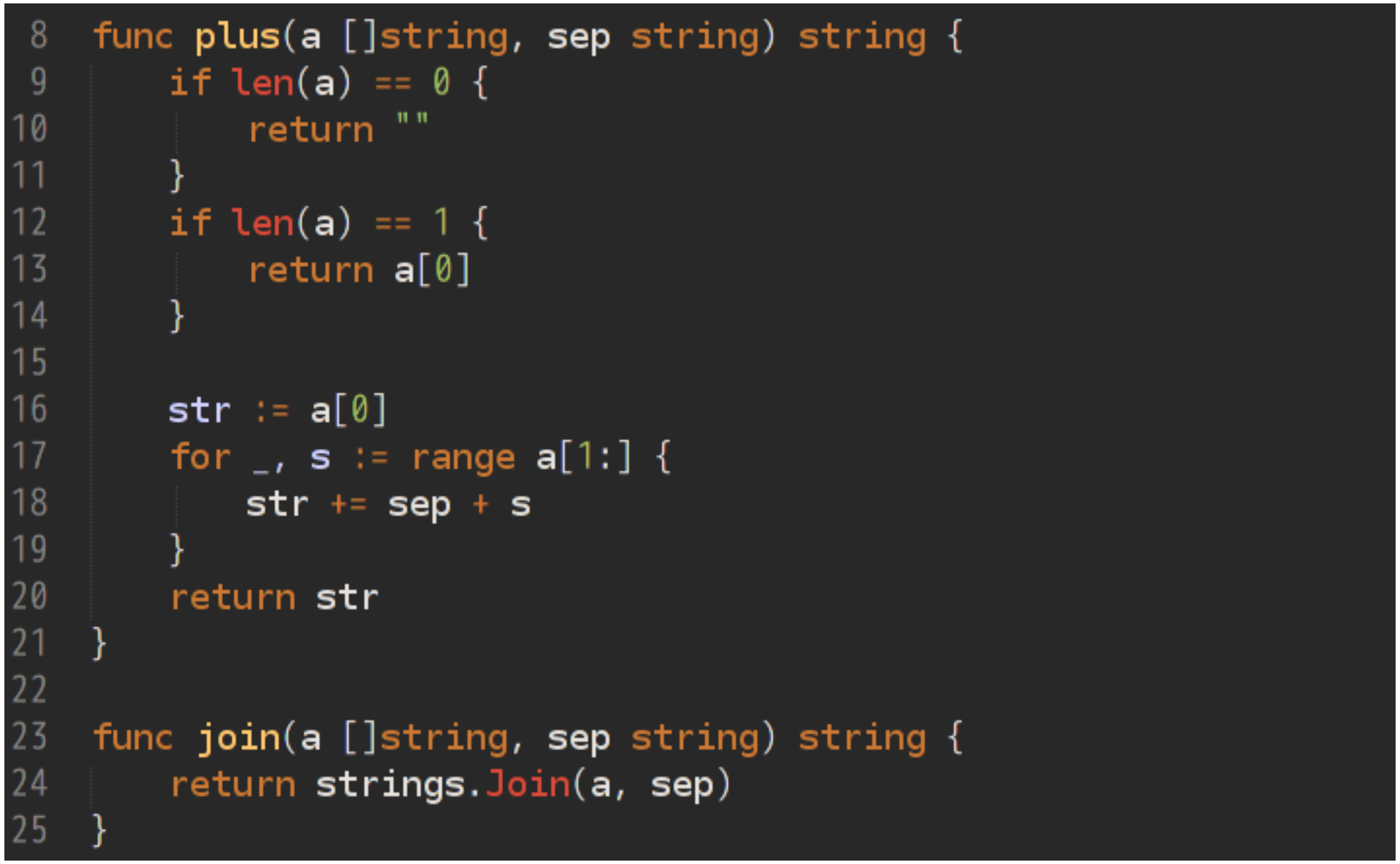


```go
    intA int   = 12345
    intB int64 = 67890
    str        = "hello gohpers!"
    sep        = ","
)

func BenchmarkFmt(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = fmt.Sprintf("%d%s%s%s%d", intA, sep, str, sep, intB)
    }
}

func BenchmarkPlus(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = strconv.Itoa(intA) + sep + str + sep + strconv.FormatInt(intB, 10)
    }
}
```

```
BenchmarkFmt      1000000      1324 ns/op
BenchmarkPlus     5000000       751 ns/op
```

string 连接3

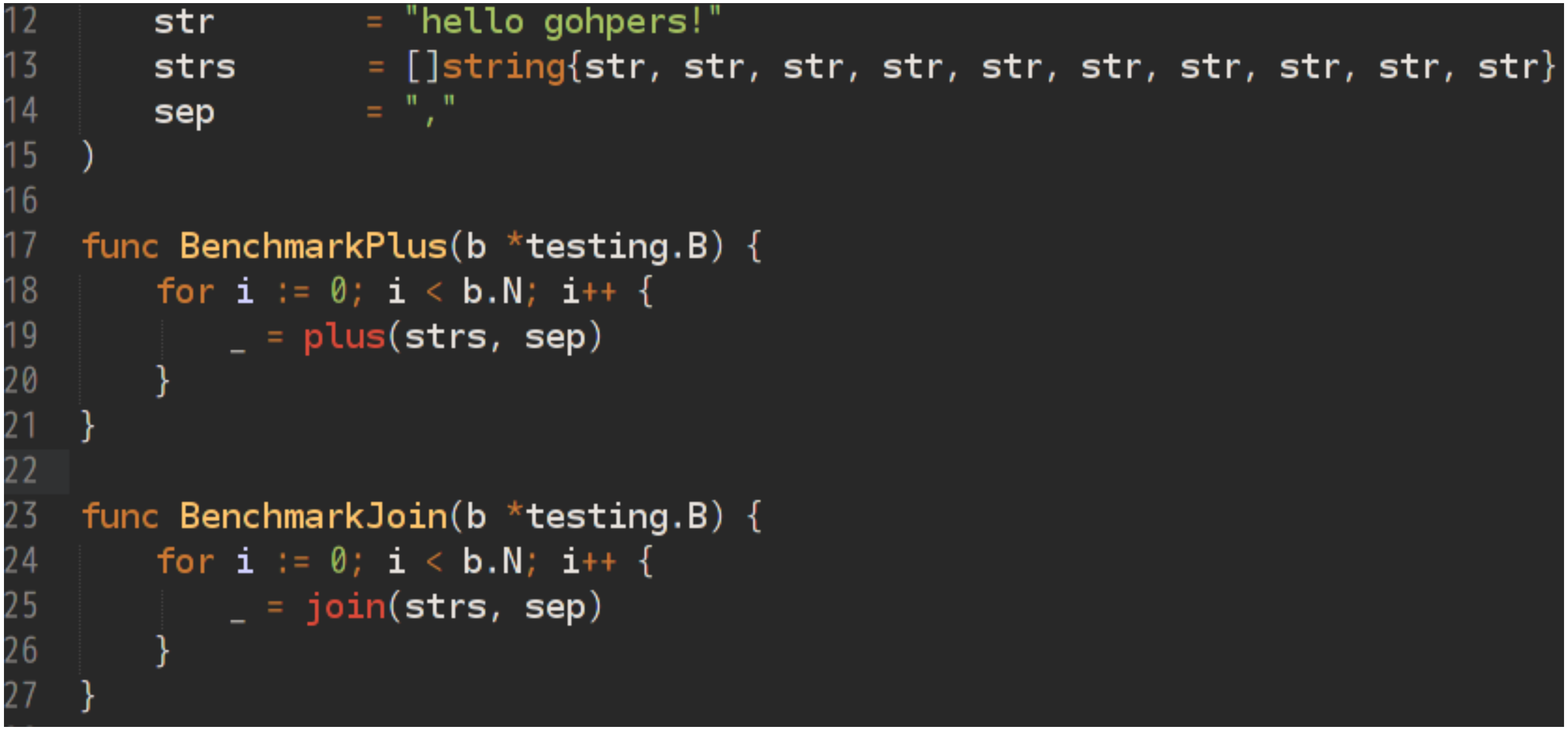
strings.join VS "+"



```go
func plus(a []string, sep string) string {
    if len(a) == 0 {
        return ""
    }
    if len(a) == 1 {
        return a[0]
    }

    str := a[0]
    for _, s := range a[1:] {
        str += sep + s
    }
    return str
}

func join(a []string, sep string) string {
    return strings.Join(a, sep)
}
```

string 连接3

strings.join VS "+"

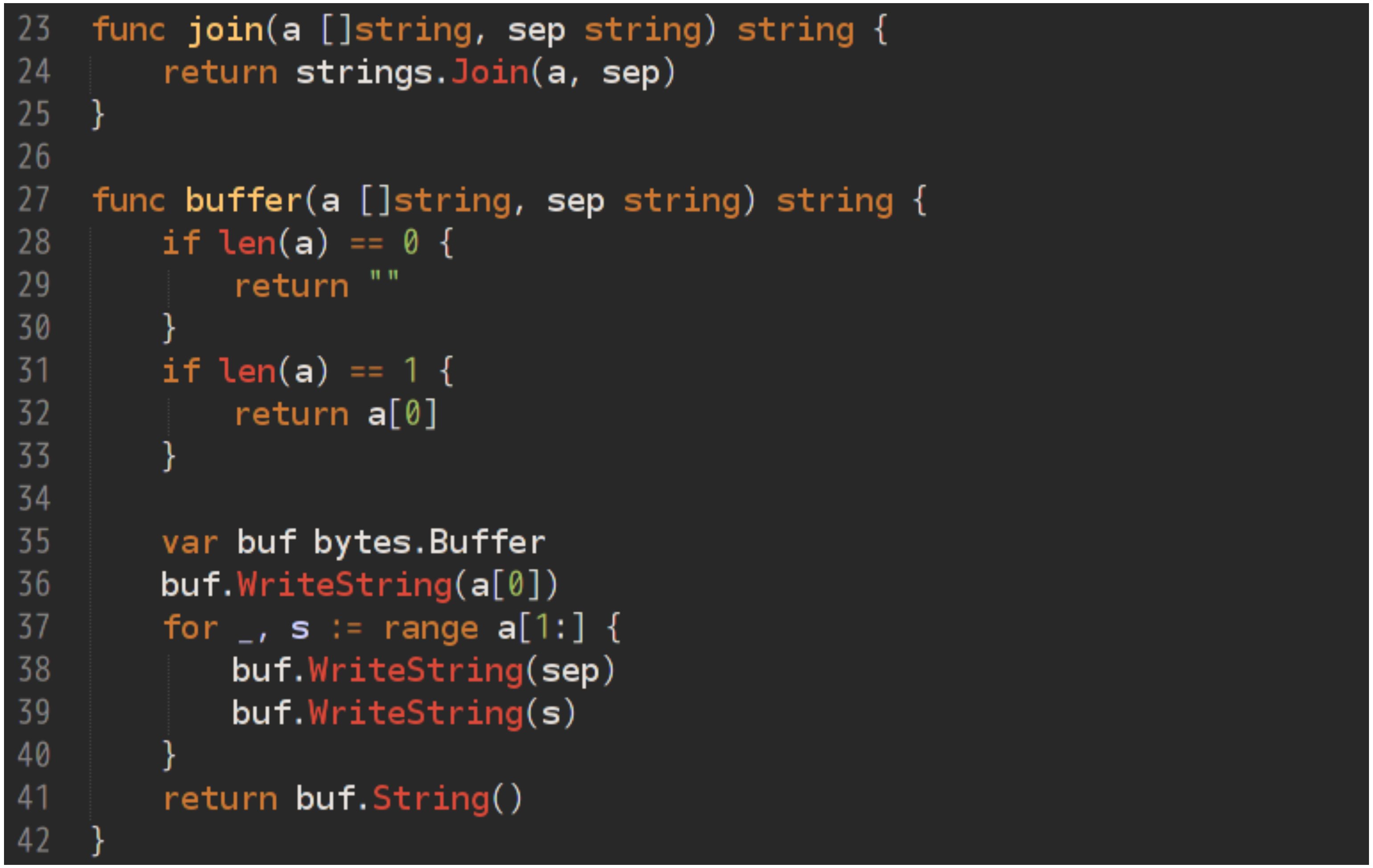


```go
	str         = "hello gohpers!"
	strs        = []string{str, str, str, str, str, str, str, str, str, str}
	sep         = ","
)

func BenchmarkPlus(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = plus(strs, sep)
	}
}

func BenchmarkJoin(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = join(strs, sep)
	}
}
```

```
BenchmarkPlus     500000      4659 ns/op
BenchmarkJoin     1000000     1491 ns/op
```

string 连接4

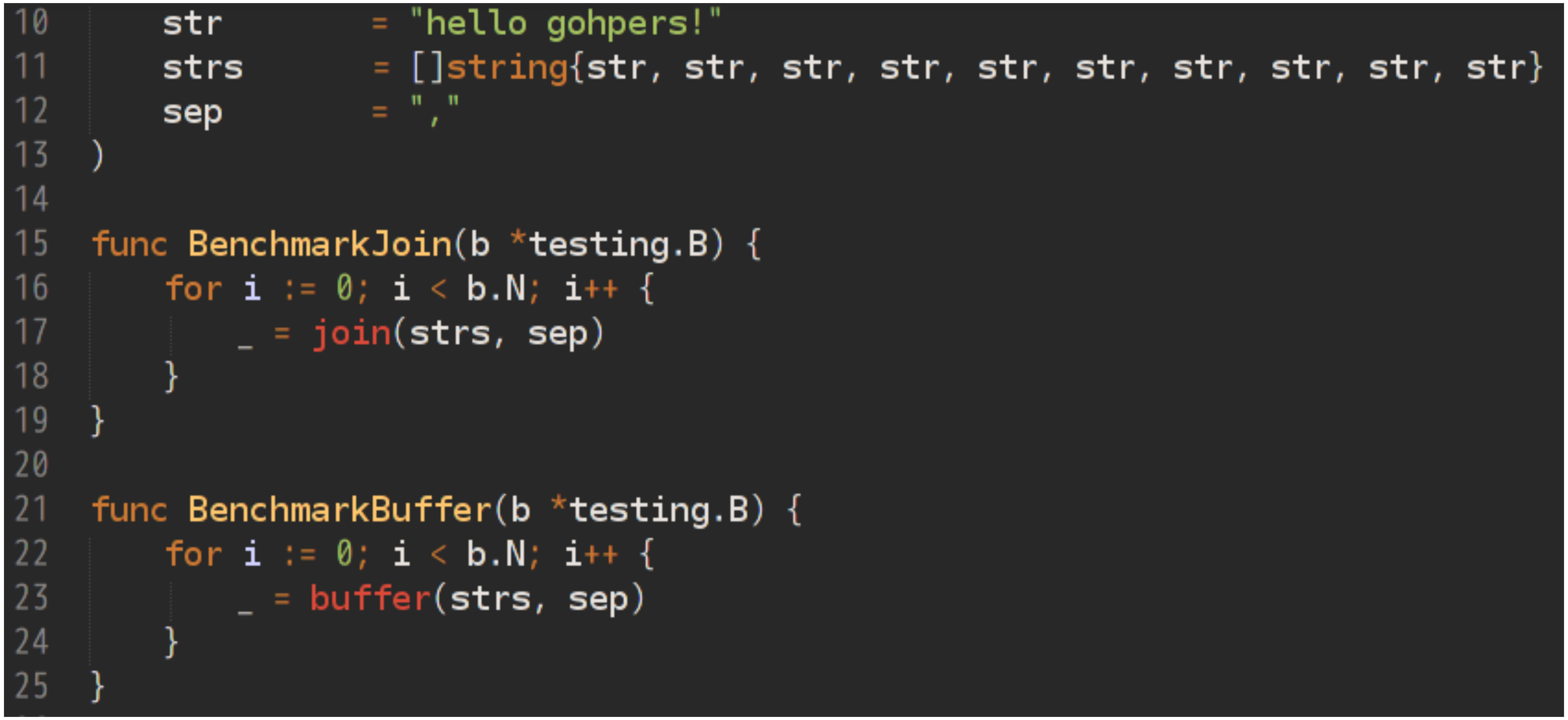
strings.Join VS bytes.Buffer



```go
func join(a []string, sep string) string {
    return strings.Join(a, sep)
}

func buffer(a []string, sep string) string {
    if len(a) == 0 {
        return ""
    }
    if len(a) == 1 {
        return a[0]
    }

    var buf bytes.Buffer
    buf.WriteString(a[0])
    for _, s := range a[1:] {
        buf.WriteString(sep)
        buf.WriteString(s)
    }
    return buf.String()
}
```

string 连接4

strings.Join VS bytes.Buffer

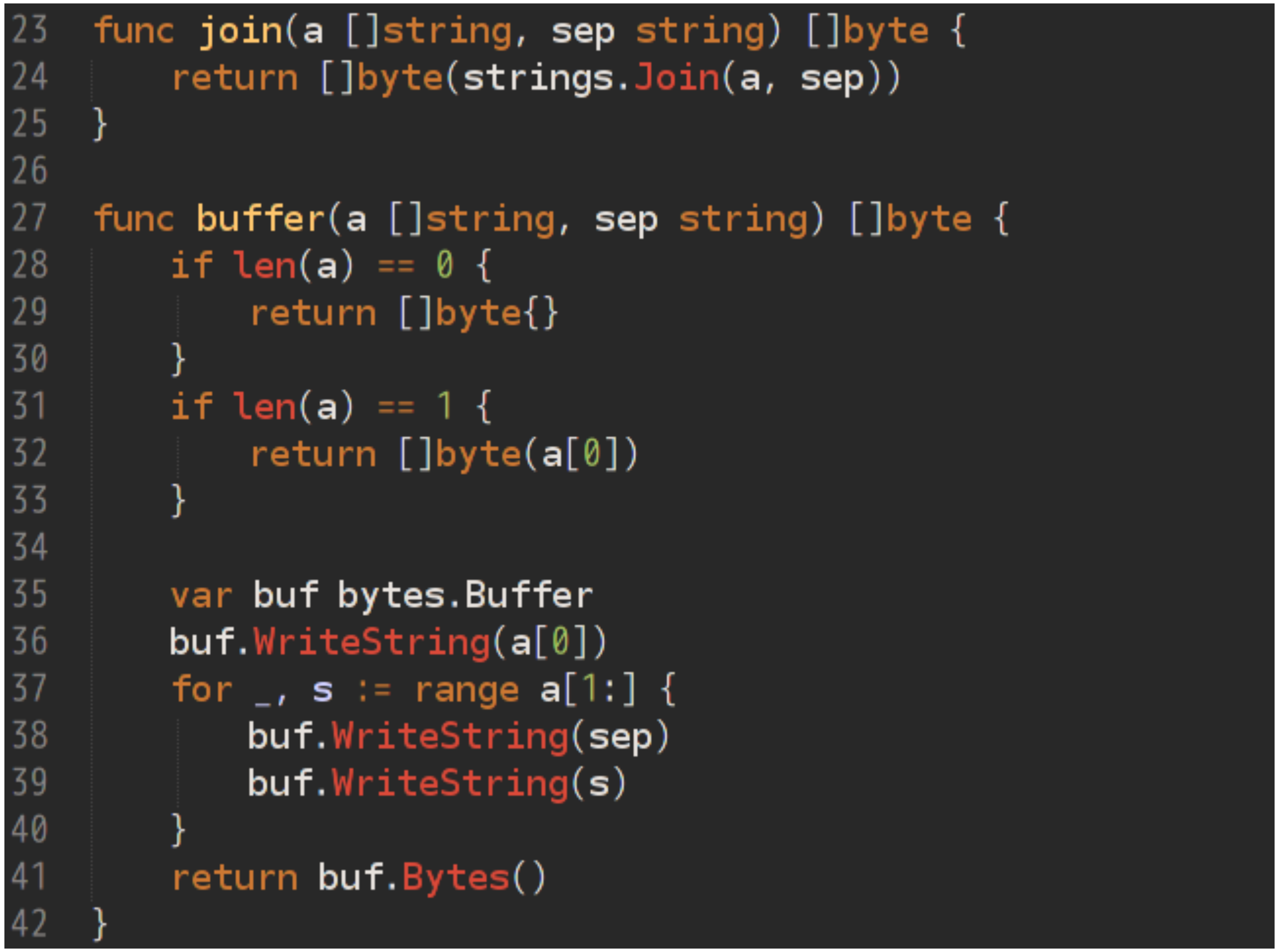


```go
	str         = "hello gohpers!"
	strs        = []string{str, str, str, str, str, str, str, str, str, str}
	sep         = ","
)

func BenchmarkJoin(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = join(strs, sep)
	}
}

func BenchmarkBuffer(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = buffer(strs, sep)
	}
}
```

BenchmarkJoin	1000000	1505 ns/op
BenchmarkBuffer	500000	2886 ns/op

string 连接4-1

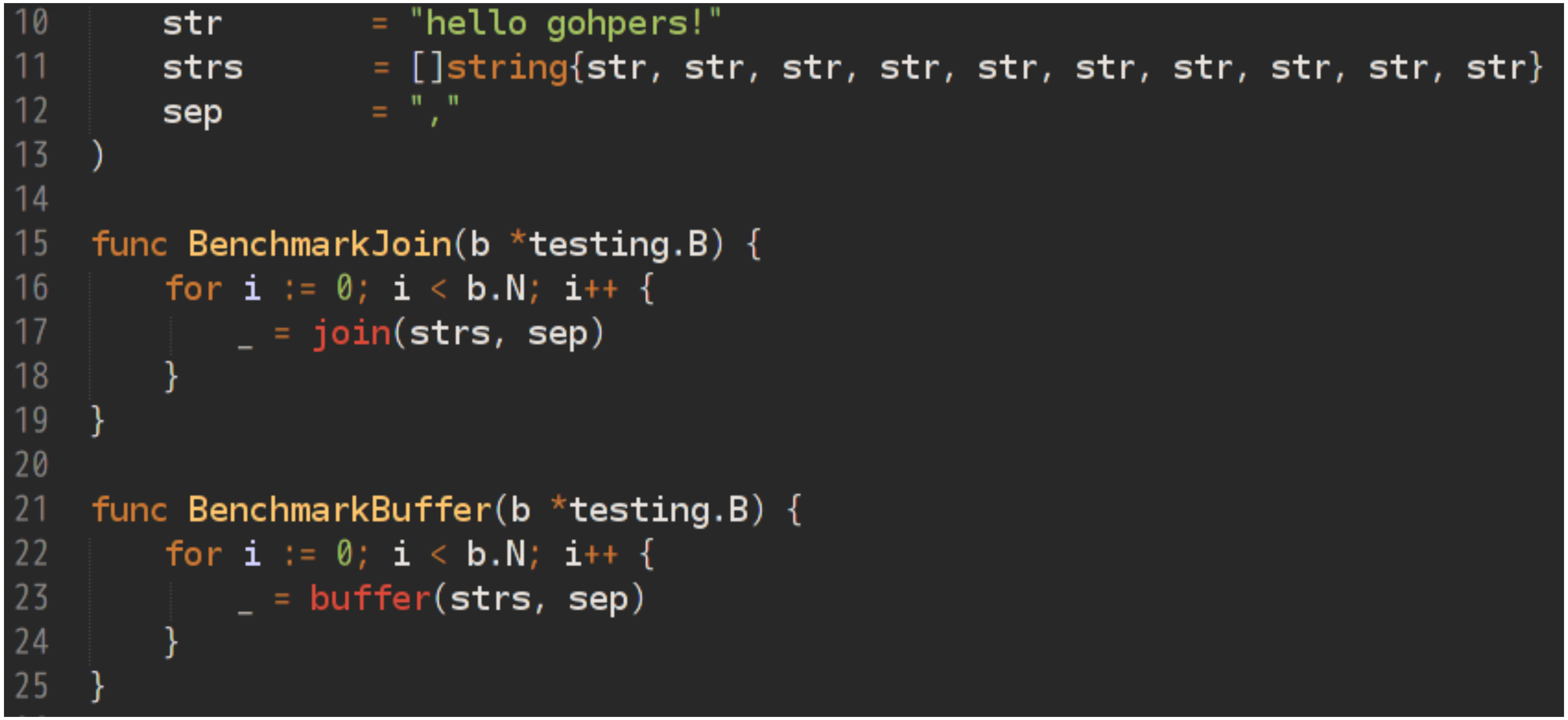
strings.Join VS bytes.Buffer



```go
func join(a []string, sep string) []byte {
    return []byte(strings.Join(a, sep))
}

func buffer(a []string, sep string) []byte {
    if len(a) == 0 {
        return []byte{}
    }
    if len(a) == 1 {
        return []byte(a[0])
    }

    var buf bytes.Buffer
    buf.WriteString(a[0])
    for _, s := range a[1:] {
        buf.WriteString(sep)
        buf.WriteString(s)
    }
    return buf.Bytes()
}
```

string 连接4-1

strings.Join VS bytes.Buffer



```go
	str          = "hello gohpers!"
	strs         = []string{str, str, str, str, str, str, str, str, str, str}
	sep          = ","
)

func BenchmarkJoin(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = join(strs, sep)
	}
}

func BenchmarkBuffer(b *testing.B) {
	for i := 0; i < b.N; i++ {
		_ = buffer(strs, sep)
	}
}
```

```
BenchmarkJoin      1000000       1824 ns/op
BenchmarkBuffer  1000000       2588 ns/op
```

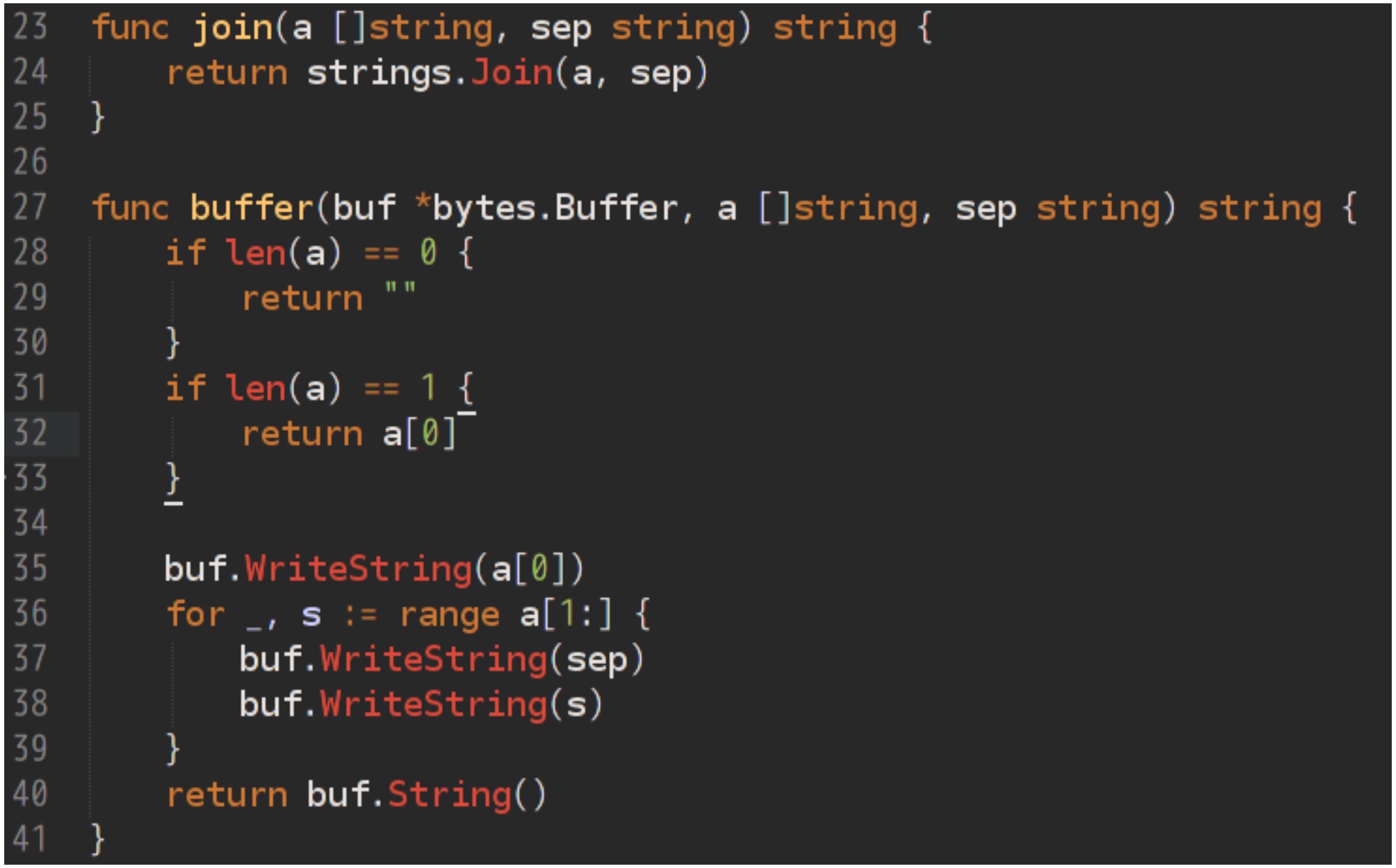
内个...内个，我对**bytes.Buffer**
情有独钟，能不能让Ta快点？

profiling

- go test -c
- go test -test.bench=. -test.cpuprofile=cpu.prof
- go tool pprof bench.test cpu.prof

string 连接4-2

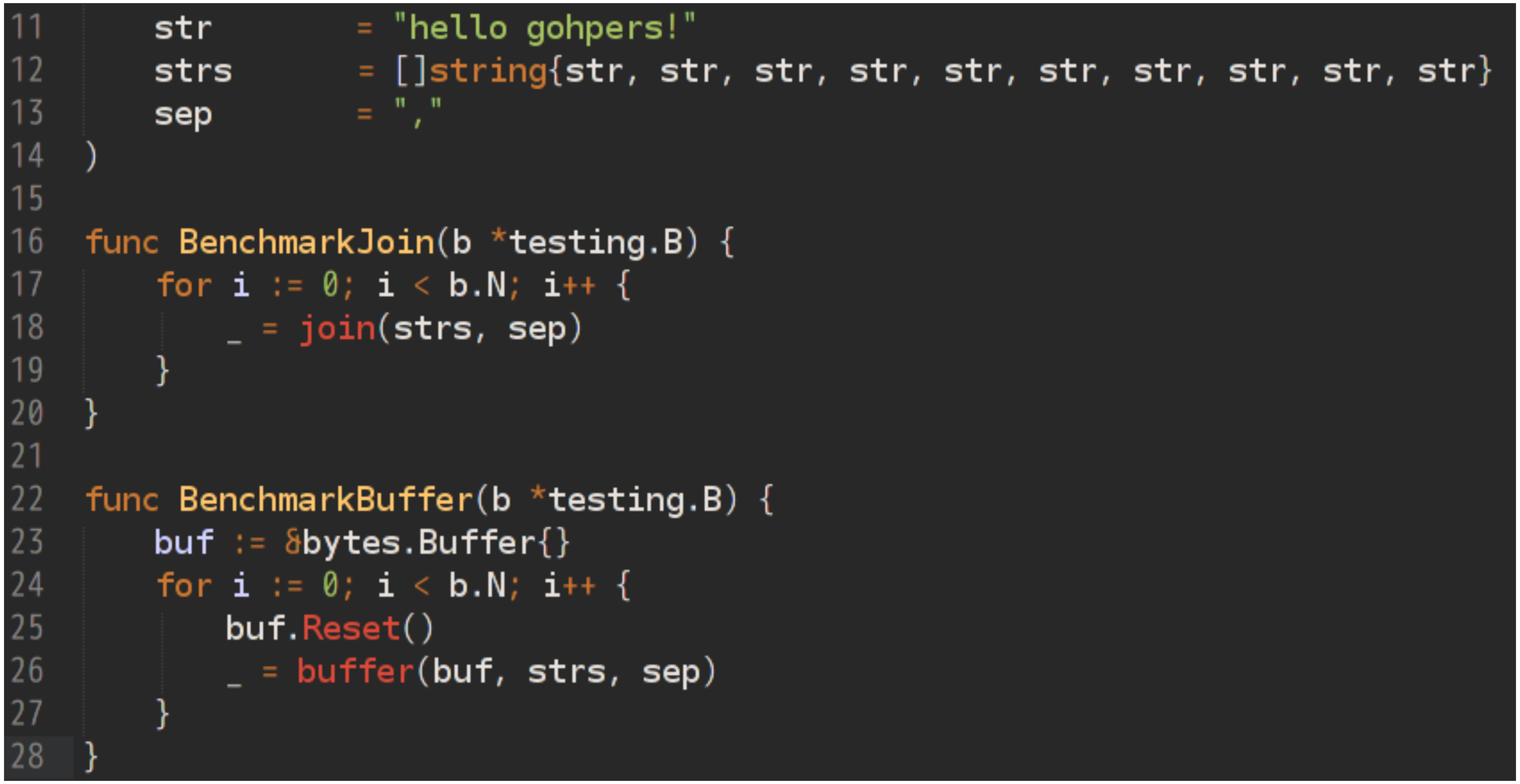
strings.Join VS bytes.Buffer



```go
func join(a []string, sep string) string {
	return strings.Join(a, sep)
}

func buffer(buf *bytes.Buffer, a []string, sep string) string {
	if len(a) == 0 {
		return ""
	}
	if len(a) == 1 {
		return a[0]
	}

	buf.WriteString(a[0])
	for _, s := range a[1:] {
		buf.WriteString(sep)
		buf.WriteString(s)
	}
	return buf.String()
}
```

string 连接4-2

strings.Join VS bytes.Buffer

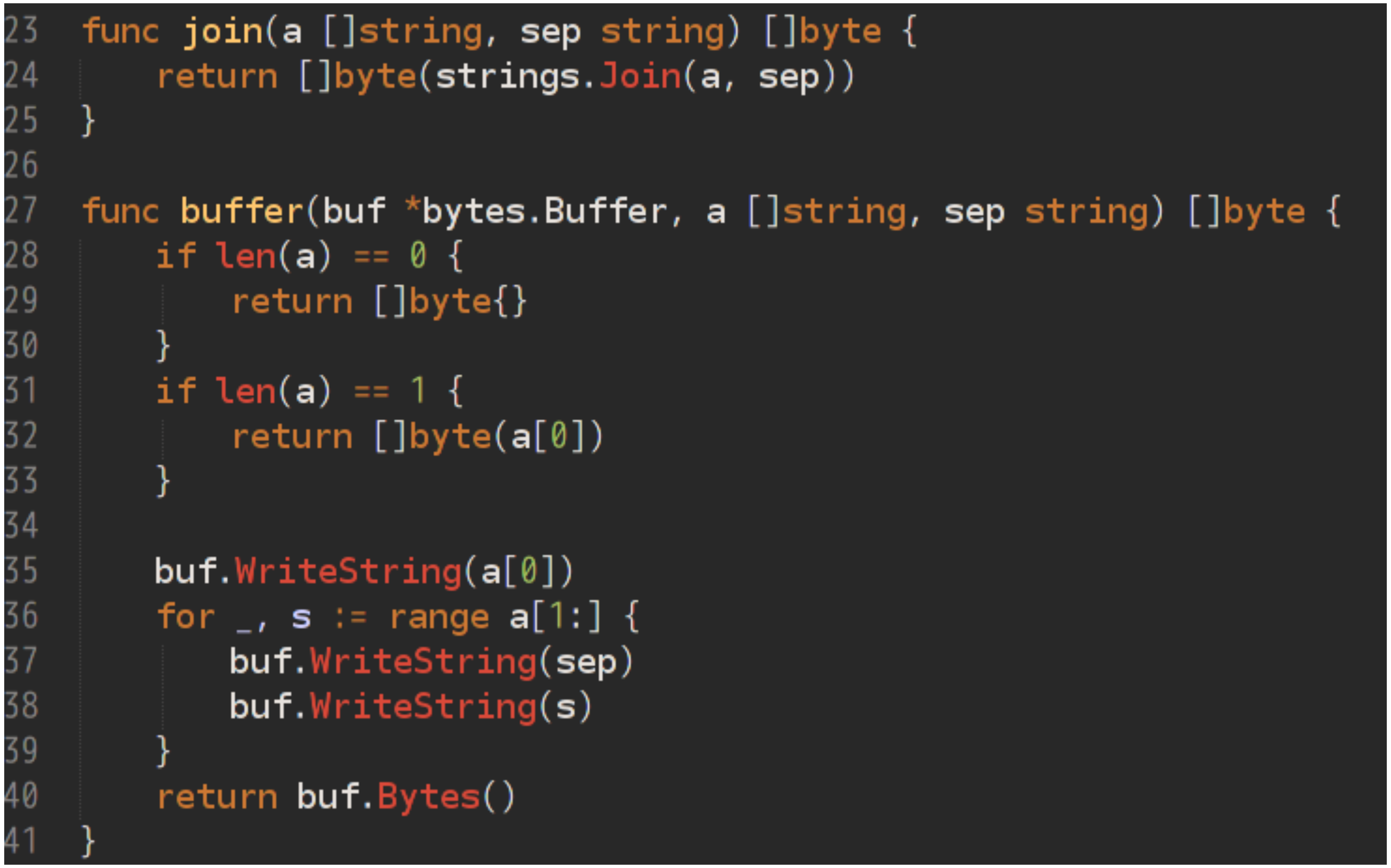


```
    str         = "hello gohpers!"
    strs        = []string{str, str, str, str, str, str, str, str, str, str}
    sep         = ","
)

func BenchmarkJoin(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = join(strs, sep)
    }
}

func BenchmarkBuffer(b *testing.B) {
    buf := &bytes.Buffer{}
    for i := 0; i < b.N; i++ {
        buf.Reset()
        _ = buffer(buf, strs, sep)
    }
}
```

```
BenchmarkJoin      1000000        1500 ns/op
BenchmarkBuffer    1000000        1482 ns/op
```

string 连接4-3

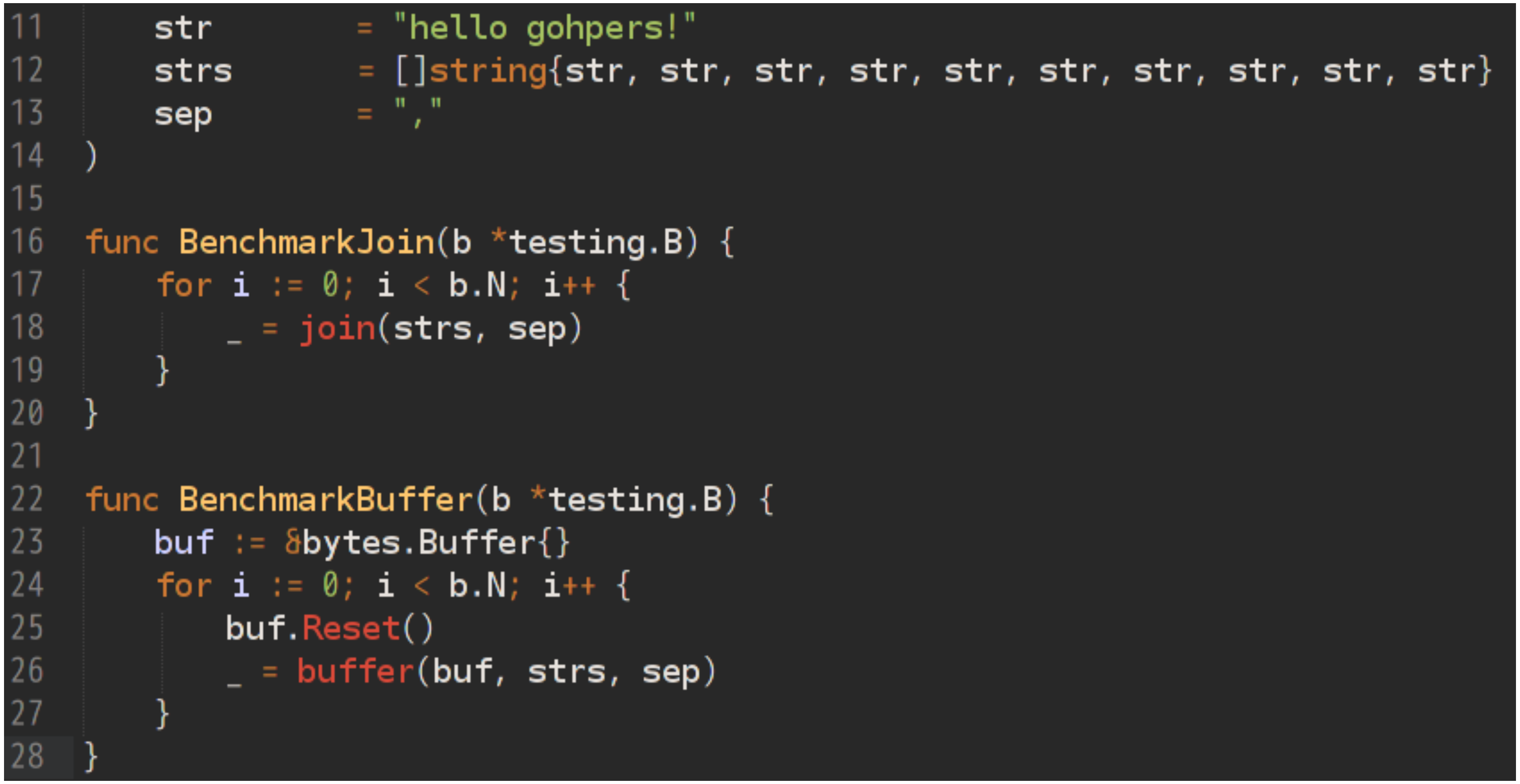
strings.Join VS bytes.Buffer



```go
func join(a []string, sep string) []byte {
    return []byte(strings.Join(a, sep))
}

func buffer(buf *bytes.Buffer, a []string, sep string) []byte {
    if len(a) == 0 {
        return []byte{}
    }
    if len(a) == 1 {
        return []byte(a[0])
    }

    buf.WriteString(a[0])
    for _, s := range a[1:] {
        buf.WriteString(sep)
        buf.WriteString(s)
    }
    return buf.Bytes()
}
```

string 连接4-3

strings.Join VS bytes.Buffer



```go
    str          = "hello gohpers!"
    strs         = []string{str, str, str, str, str, str, str, str, str, str}
    sep          = ","
)

func BenchmarkJoin(b *testing.B) {
    for i := 0; i < b.N; i++ {
        _ = join(strs, sep)
    }
}

func BenchmarkBuffer(b *testing.B) {
    buf := &bytes.Buffer{}
    for i := 0; i < b.N; i++ {
        buf.Reset()
        _ = buffer(buf, strs, sep)
    }
}
```

```
BenchmarkJoin       1000000         1791 ns/op
BenchmarkBuffer     1000000         1162 ns/op
```

string 和 []byte

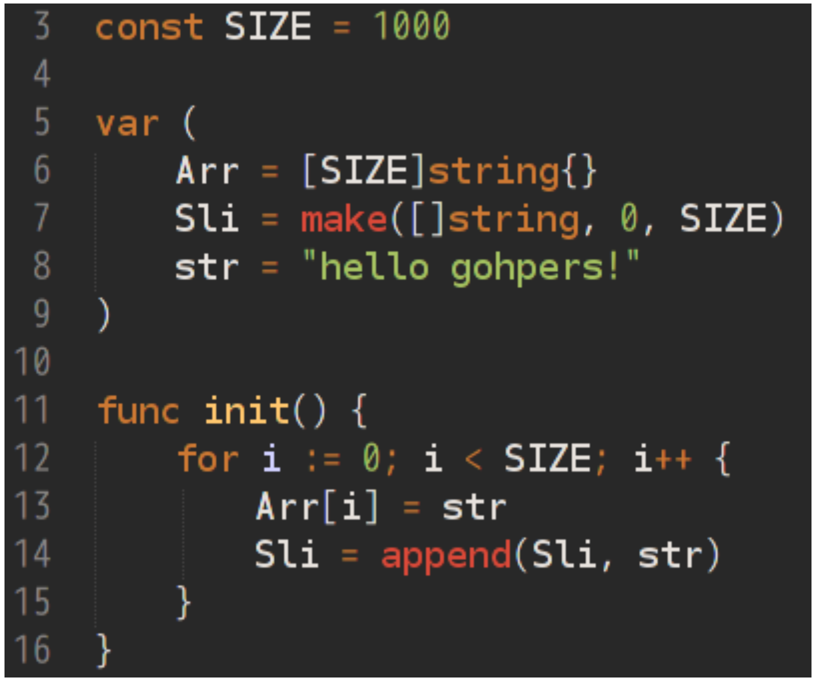
- 如果可以的话，尽量用多[]byte，少用string
- 尽可能少地在两者之间做转换
- append([]byte, string...)
- copy([]byte, string)

strconv

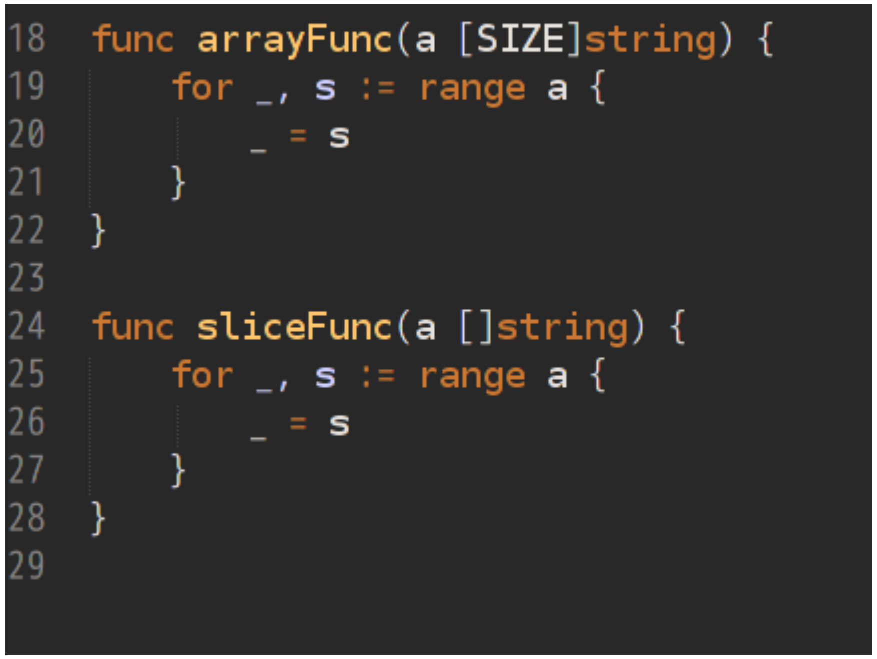
- func AppendBool(dst []byte, b bool) []byte
- func AppendFloat(dst []byte, f float64, fmt byte, prec int, bitSize int) []byte
- func AppendInt(dst []byte, i int64, base int) []byte
- func AppendUint(dst []byte, i uint64, base int) []byte
- func FormatBool(b bool) string
- func FormatFloat(f float64, fmt byte, prec, bitSize int) string
- func FormatInt(i int64, base int) string
- func FormatUint(i uint64, base int) string

slice & array

slice & array

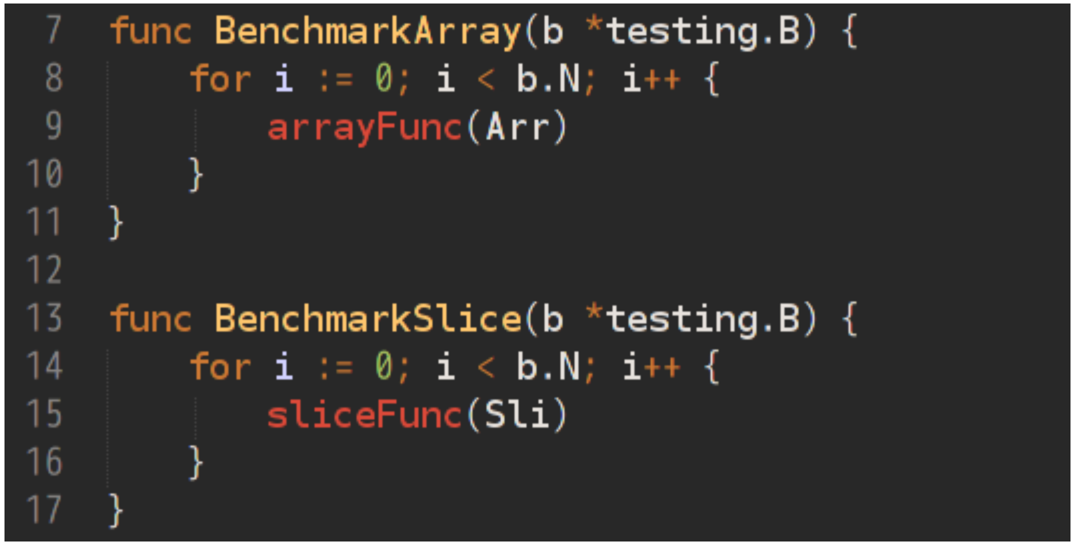


```go
const SIZE = 1000

var (
	Arr = [SIZE]string{}
	Sli = make([]string, 0, SIZE)
	str = "hello gohpers!"
)

func init() {
	for i := 0; i < SIZE; i++ {
		Arr[i] = str
		Sli = append(Sli, str)
	}
}
```



```go
func arrayFunc(a [SIZE]string) {
	for _, s := range a {
		_ = s
	}
}

func sliceFunc(a []string) {
	for _, s := range a {
		_ = s
	}
}

```

slice & array



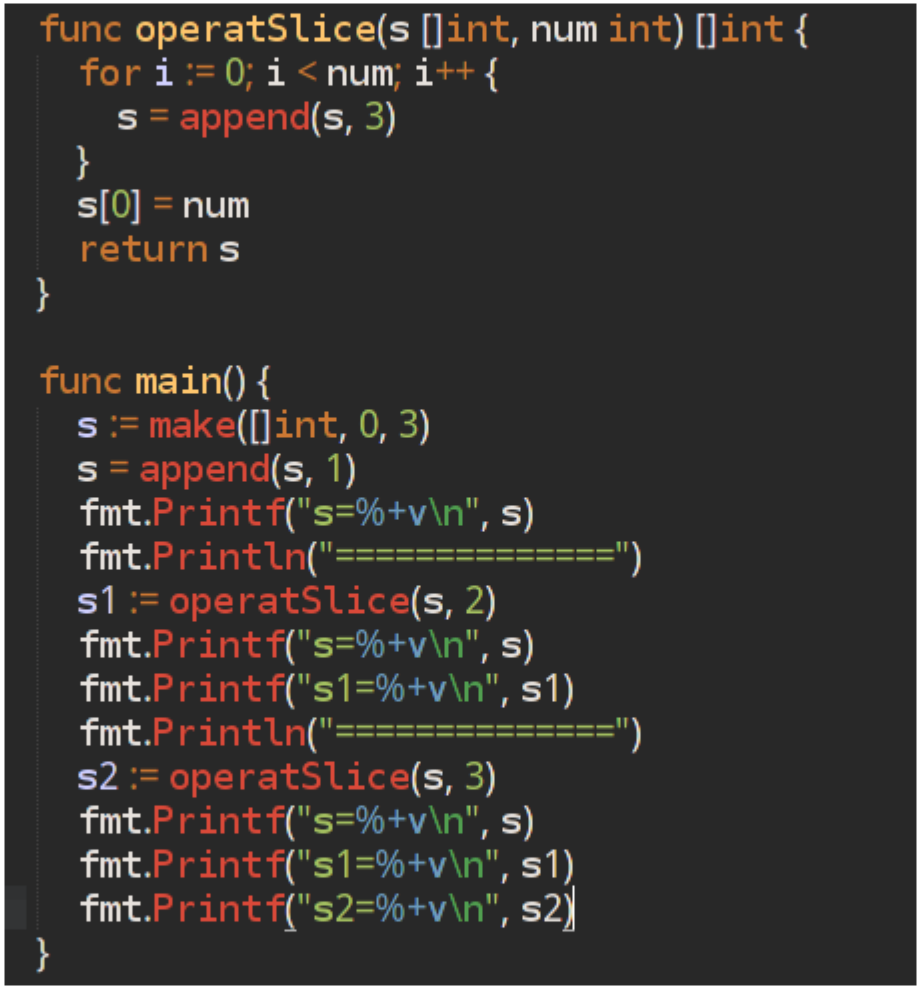
```go
func BenchmarkArray(b *testing.B) {
	for i := 0; i < b.N; i++ {
		arrayFunc(Arr)
	}
}

func BenchmarkSlice(b *testing.B) {
	for i := 0; i < b.N; i++ {
		sliceFunc(Sli)
	}
}
```

```
BenchmarkArray     200000     11101 ns/op
BenchmarkSlice    2000000       822 ns/op
```

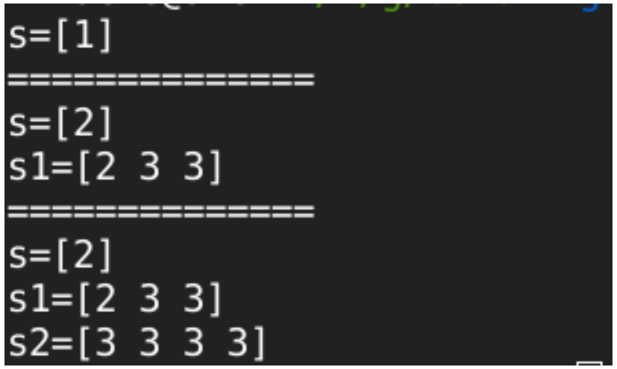
slice & array

- 数组是值传递
- slice是引用传递

Slice 坑

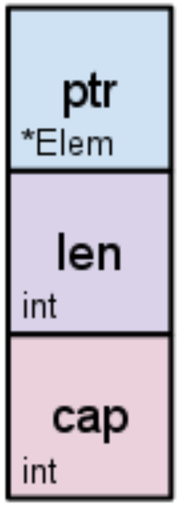


```go
func operatSlice(s []int, num int) []int {
    for i := 0; i < num; i++ {
        s = append(s, 3)
    }
    s[0] = num
    return s
}

func main() {
    s := make([]int, 0, 3)
    s = append(s, 1)
    fmt.Printf("s=%+v\n", s)
    fmt.Println("==============")
    s1 := operatSlice(s, 2)
    fmt.Printf("s=%+v\n", s)
    fmt.Printf("s1=%+v\n", s1)
    fmt.Println("==============")
    s2 := operatSlice(s, 3)
    fmt.Printf("s=%+v\n", s)
    fmt.Printf("s1=%+v\n", s1)
    fmt.Printf("s2=%+v\n", s2)
}
```

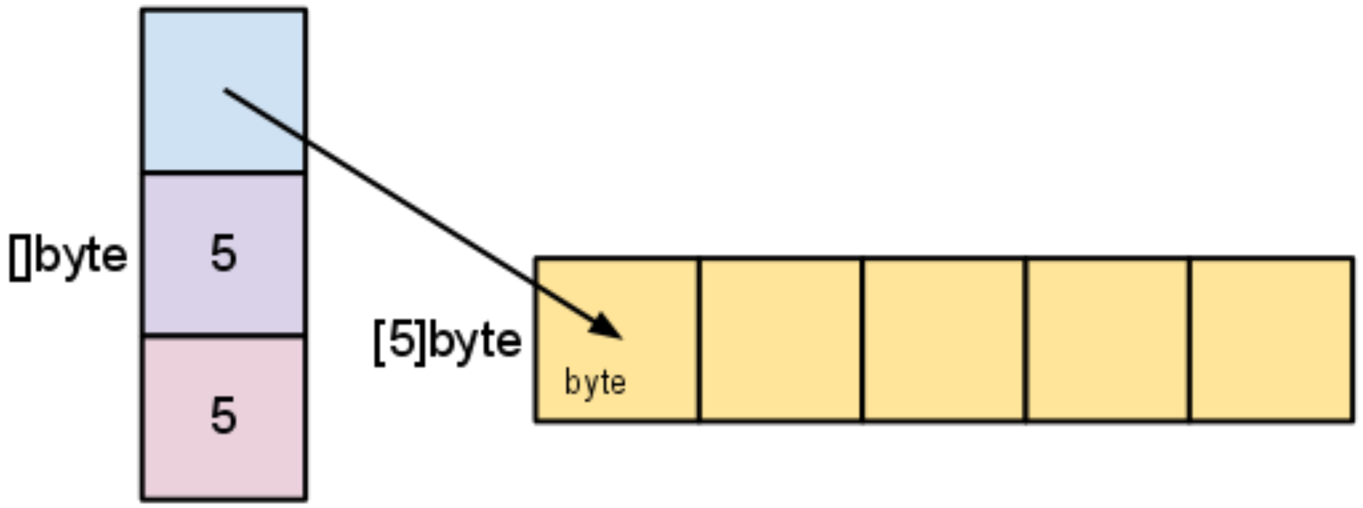


```
s=[1]
==============
s=[2]
s1=[2 3 3]
==============
s=[2]
s1=[2 3 3]
s2=[3 3 3 3]
```

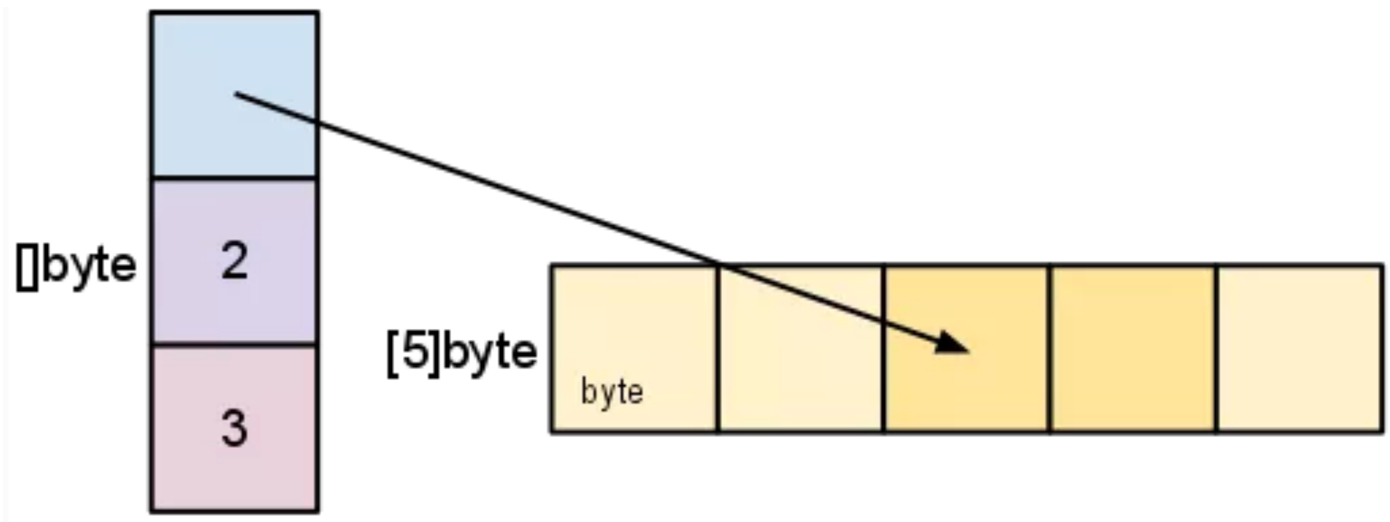
slice 结构



make([]byte, 5)

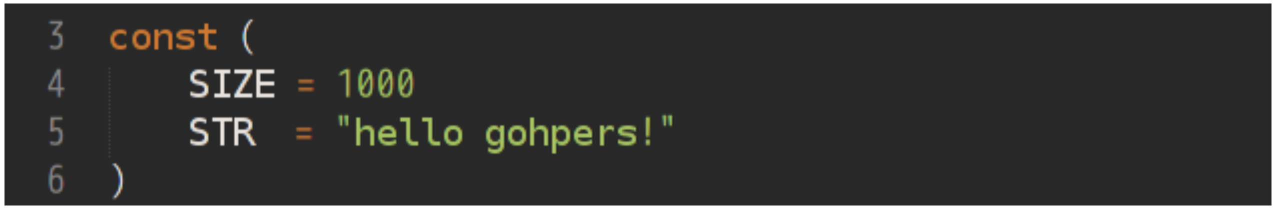


s = s[2:4]

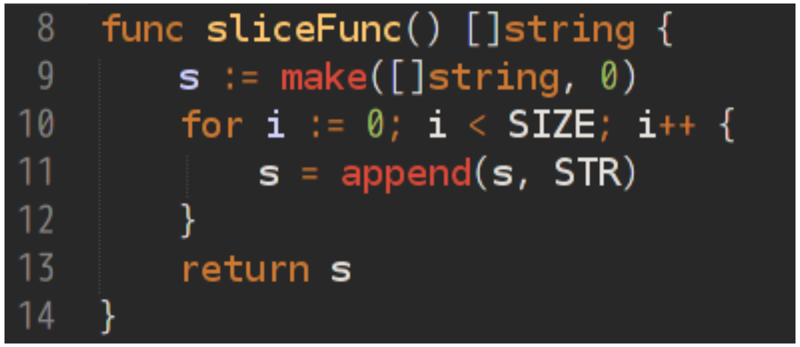


slice & map 初始化

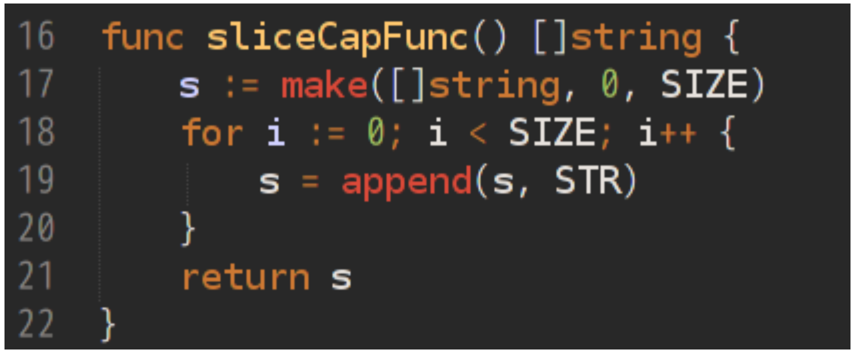
slice 初始化



```go
const (
    SIZE = 1000
    STR  = "hello gohpers!"
)
```

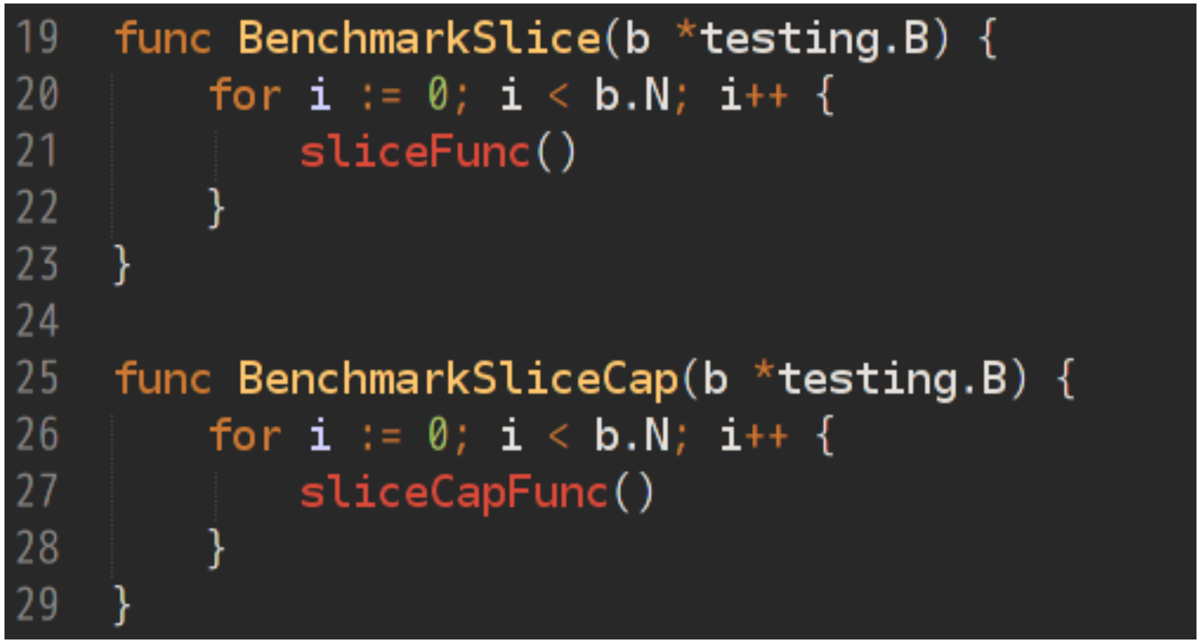


```go
func sliceFunc() []string {
    s := make([]string, 0)
    for i := 0; i < SIZE; i++ {
        s = append(s, STR)
    }
    return s
}
```



```go
func sliceCapFunc() []string {
    s := make([]string, 0, SIZE)
    for i := 0; i < SIZE; i++ {
        s = append(s, STR)
    }
    return s
}
```

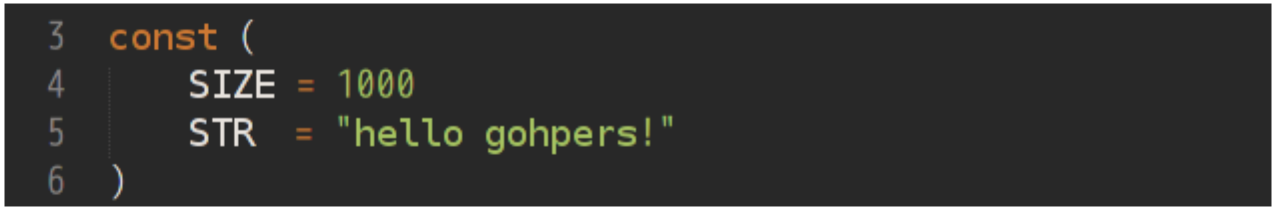
slice 初始化测试结果



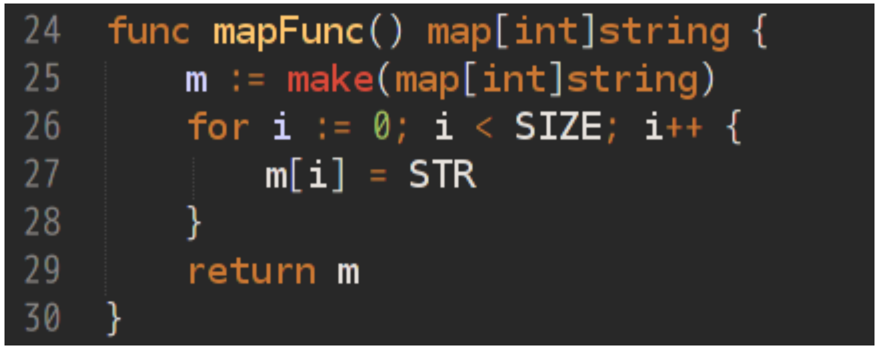
```go
func BenchmarkSlice(b *testing.B) {
	for i := 0; i < b.N; i++ {
		sliceFunc()
	}
}

func BenchmarkSliceCap(b *testing.B) {
	for i := 0; i < b.N; i++ {
		sliceCapFunc()
	}
}
```

```
BenchmarkSlice        50000     33351 ns/op
BenchmarkSliceCap     100000    16432 ns/op
```

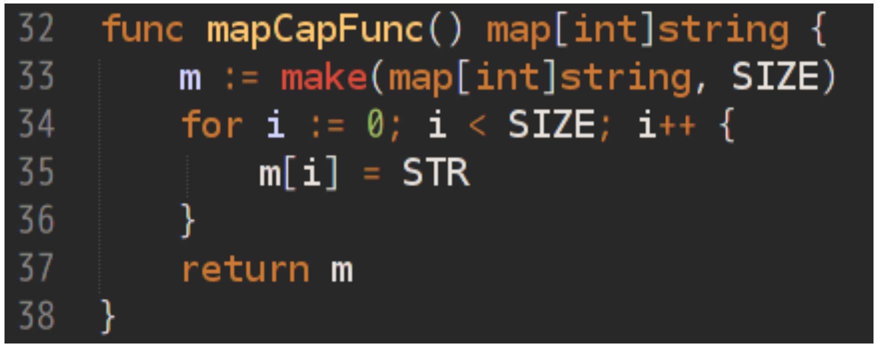
map 初始化



```go
const (
	SIZE = 1000
	STR  = "hello gohpers!"
)
```

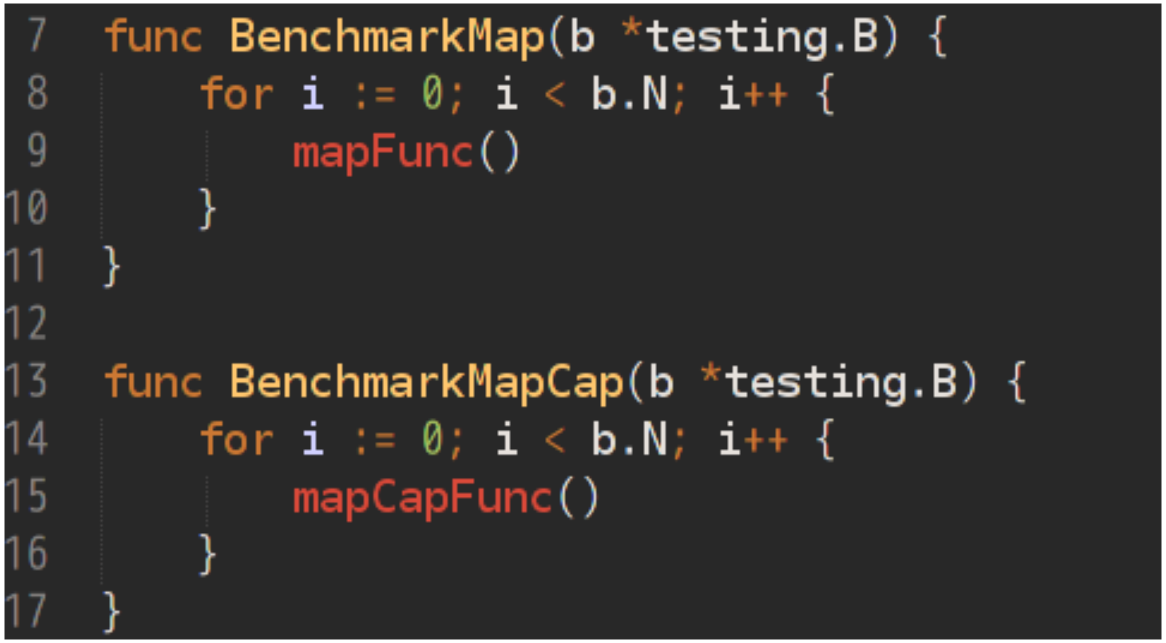


```go
func mapFunc() map[int]string {
	m := make(map[int]string)
	for i := 0; i < SIZE; i++ {
		m[i] = STR
	}
	return m
}
```



```go
func mapCapFunc() map[int]string {
	m := make(map[int]string, SIZE)
	for i := 0; i < SIZE; i++ {
		m[i] = STR
	}
	return m
}
```

map 初始化测试结果



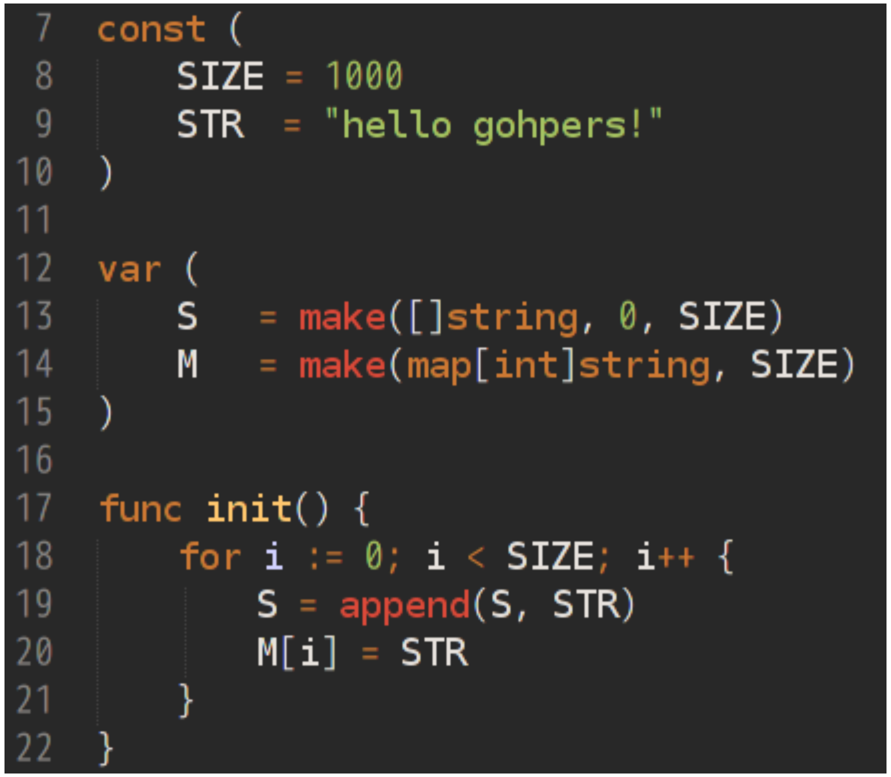
```go
func BenchmarkMap(b *testing.B) {
	for i := 0; i < b.N; i++ {
		mapFunc()
	}
}

func BenchmarkMapCap(b *testing.B) {
	for i := 0; i < b.N; i++ {
		mapCapFunc()
	}
}
```

```
BenchmarkMap            5000     277715 ns/
    op
BenchmarkMapCap        10000     136396 ns/
    op
```

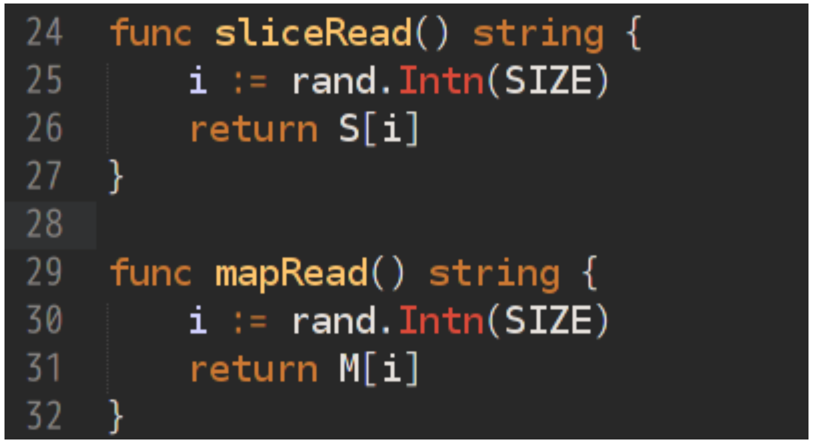
slice or map?

```
BenchmarkSlice          50000       33351 ns/op
BenchmarkMap            5000         277715 ns/op
BenchmarkSliceCap   100000      16432 ns/op
BenchmarkMapCap     10000        136396 ns/op
```

slice & map Read

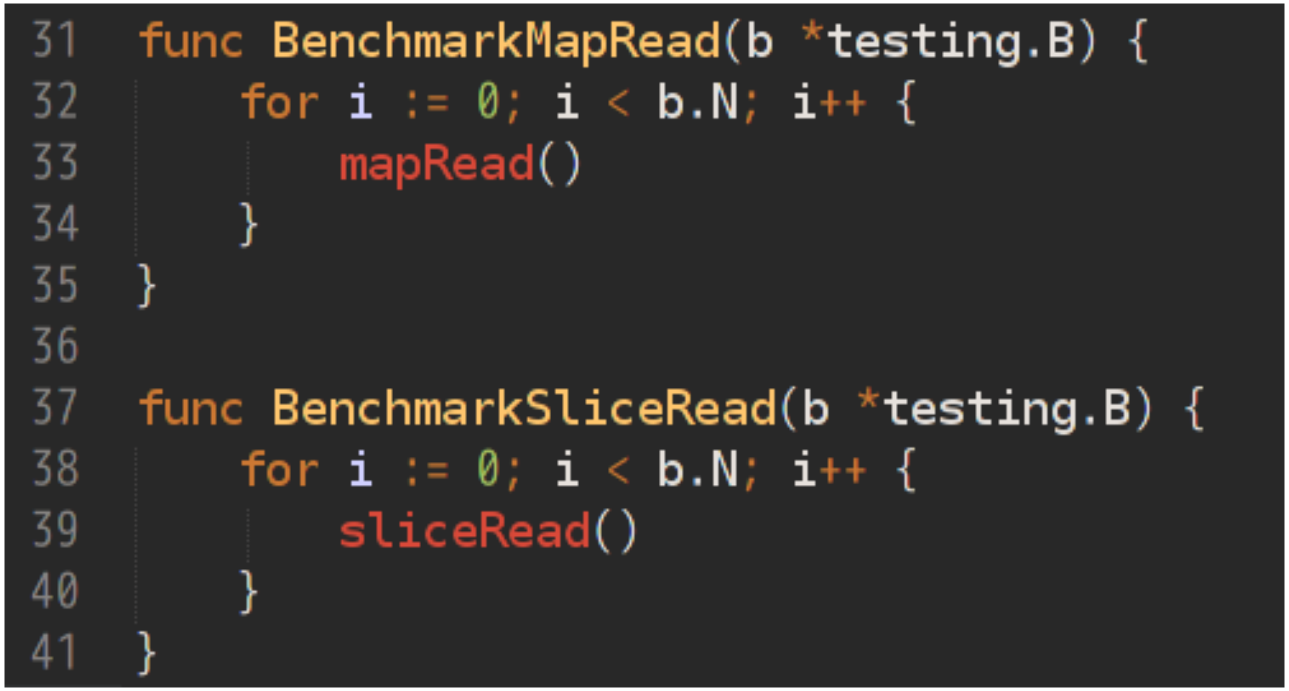


```go
const (
	SIZE = 1000
	STR  = "hello gohpers!"
)

var (
	S   = make([]string, 0, SIZE)
	M   = make(map[int]string, SIZE)
)

func init() {
	for i := 0; i < SIZE; i++ {
		S = append(S, STR)
		M[i] = STR
	}
}
```



```go
func sliceRead() string {
	i := rand.Intn(SIZE)
	return S[i]
}

func mapRead() string {
	i := rand.Intn(SIZE)
	return M[i]
}
```

slice & map Read 测试结果


```go
func BenchmarkMapRead(b *testing.B) {
    for i := 0; i < b.N; i++ {
        mapRead()
    }
}

func BenchmarkSliceRead(b *testing.B) {
    for i := 0; i < b.N; i++ {
        sliceRead()
    }
}
```

BenchmarkMapRead 10000000 155
ns/op
BenchmarkSliceRead 20000000 86.8
ns/op

并发

串行泡茶

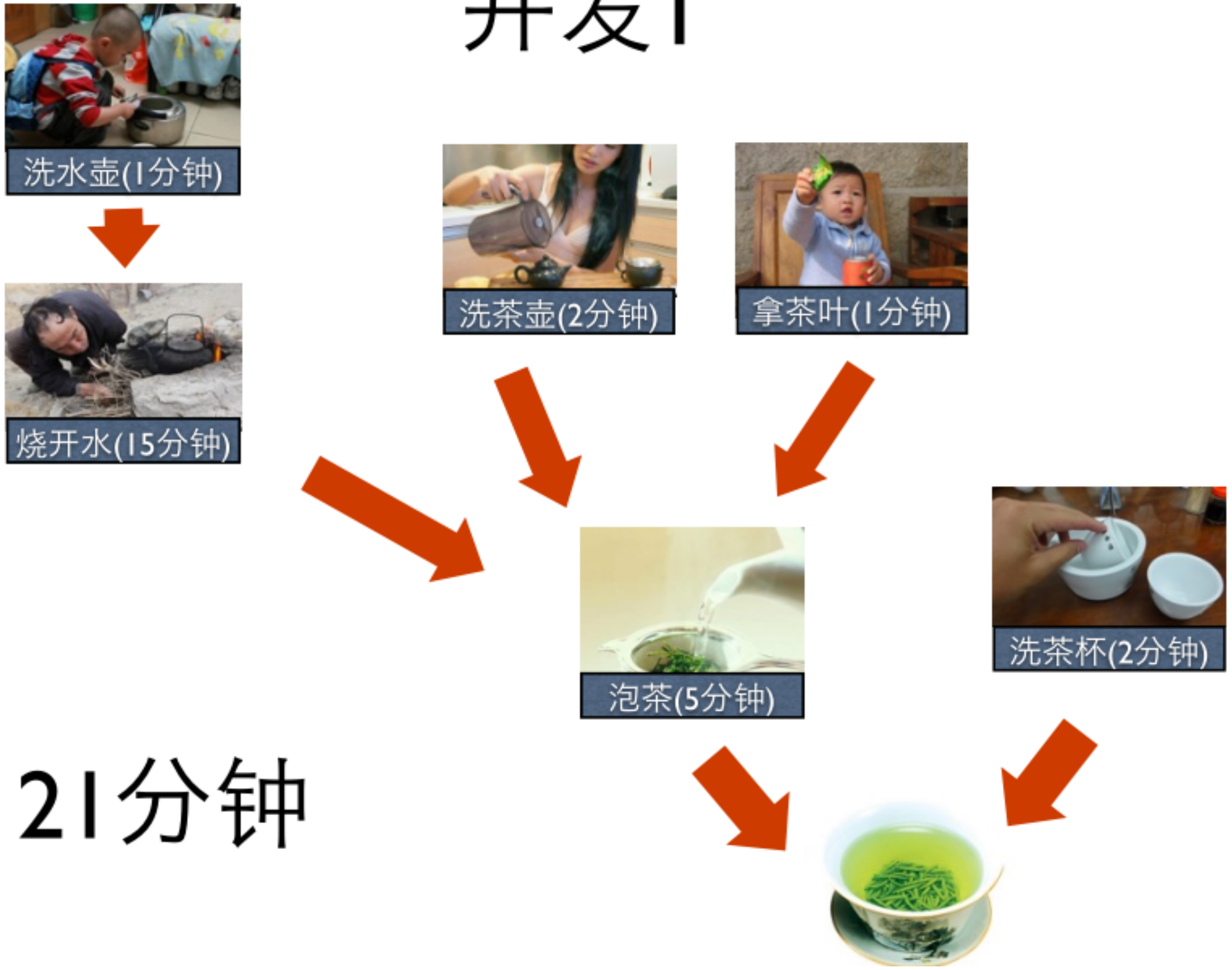
总用时 26分钟

- 洗水壶 (1分)
- 烧开水 (15分)
- 洗茶壶 (2分)
- 拿茶叶 (1分)
- 泡茶 (5分)
- 洗茶杯 (2分)

如果我要泡4杯茶？
并行

问题：
每26分钟生产一杯茶

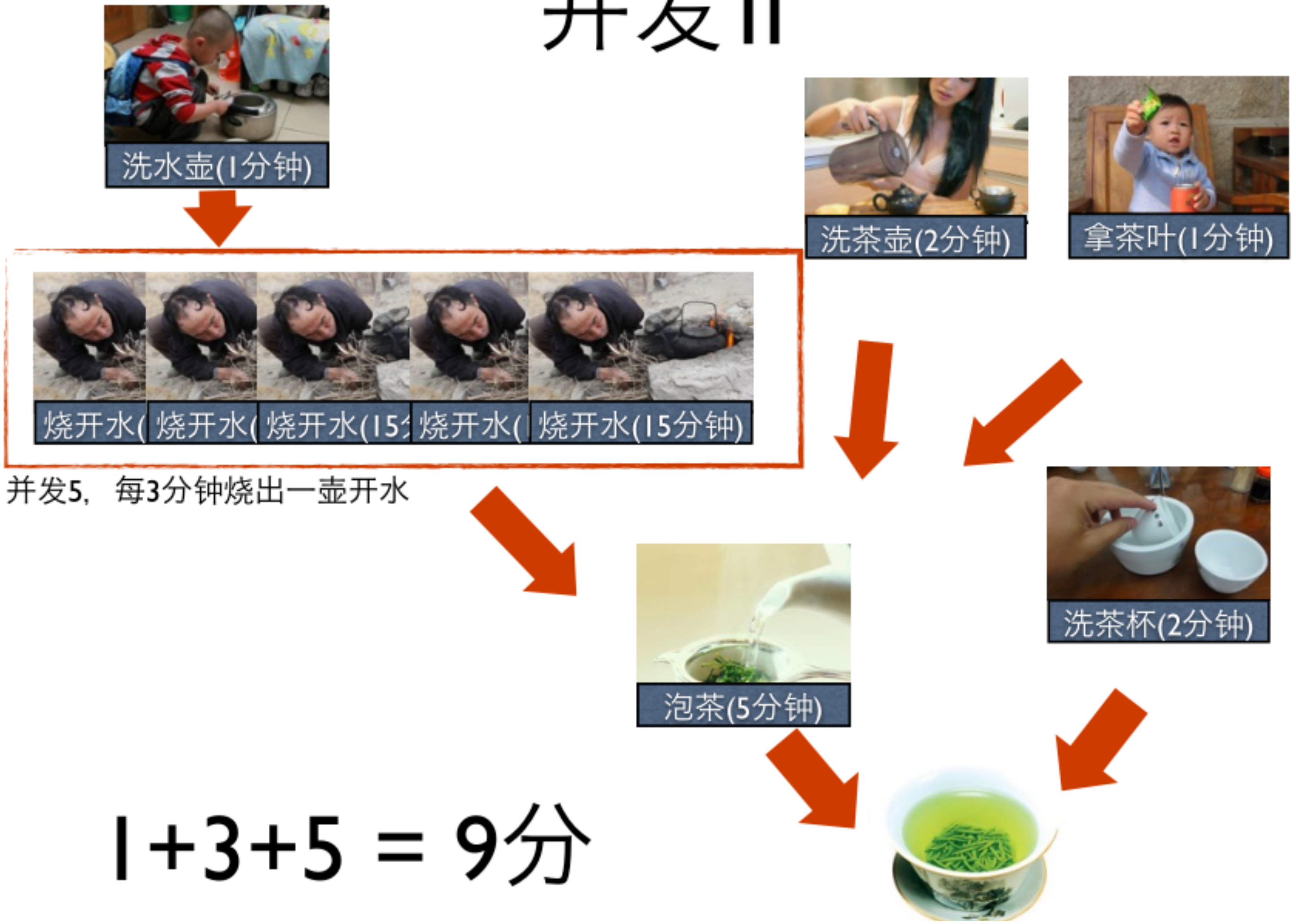
并发I



21分钟

烧开水最费时间！那么并发他！

并发II

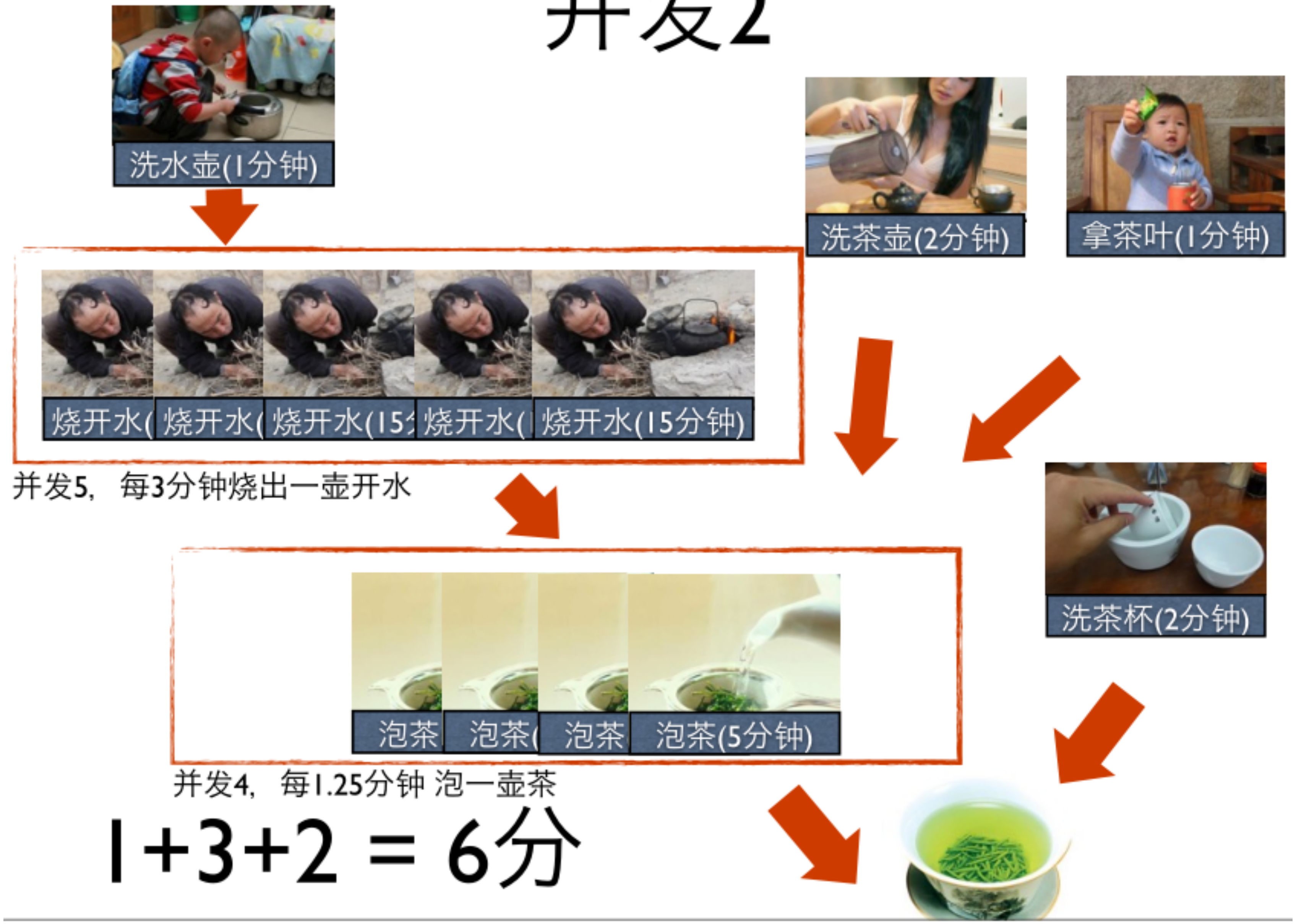


并发5， 每3分钟烧出一壶开水

1+3+5 = 9分

泡茶（5分钟）已经成为瓶颈

并发2

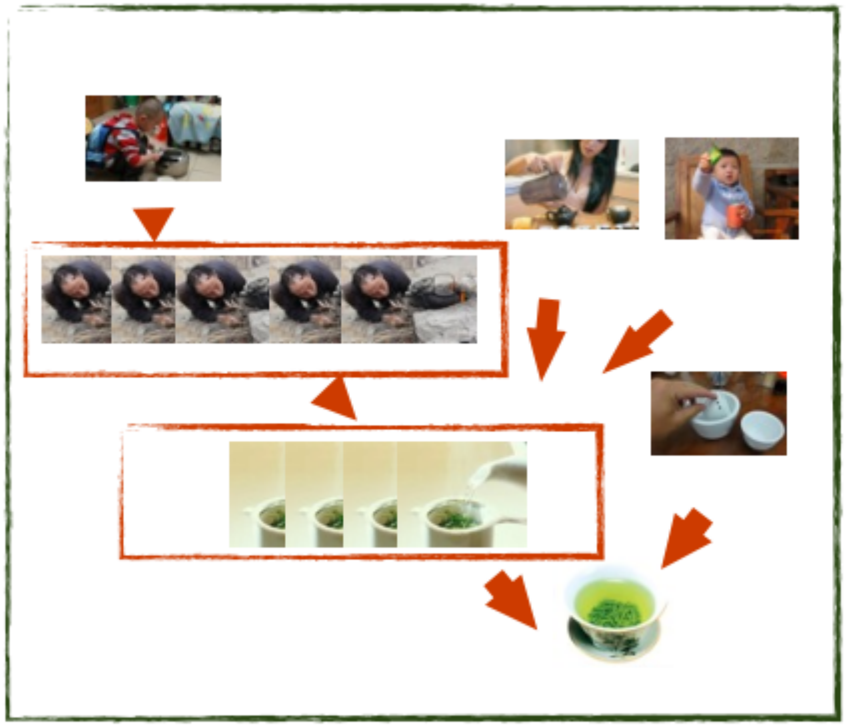


并发5， 每3分钟烧出一壶开水

并发4， 每1.25分钟 泡一壶茶

1+3+2 = 6分

并发3



每3分钟一壶

并发大于并行，包含并行

缓存

提前优化是万恶之源

Q & A

苏创绩 @miraclesu-创绩

欢迎加入我们

舜飞科技

网络运营全流程解决方案供应商