

# Go 语言的简要分析与评价

裴子祥

(北京邮电大学 计算机学院 北京 100876)

E-mail: peipeizx@outlook.com

**摘要:** 随着互联网的高速发展, 分布式系统, 区块链技术日益成熟, 每天网络上都有海量数据产生。在不断增长的业务需求中, 对于程序语言的需求更趋于简单简洁, 易开发, 高并发, 这与 Go 语言的特性不谋而合。本文就程序设计语言原理, 首先对 Go 语言在包括语言结构的简单性、规律性进行分析与评价。然后通过代码实践, Go 与 C、C++、Python、JAVA 等多种语言进行对比分析, 比较它们的执行效率, 并且对 Go 语言的并行性与它的实现机制着重分析, 比较它与 JAVA、C++ 并行程序设计模型的不同与并行化的性能提升。借此, 说明 Go 语言在简单性、规律性、效率、并行性上较现常用语言而言的优点。最后, 总结 Go 语言的特性, 提出对程序设计语言的理解与一些展望。

**关键词:** Go 语言; 简单性; 规律性; 效率; 并行性; 并行程序设计模型; 对比分析

## 1 引言

程序设计语言种类繁多, 不同的计算机应用领域可能催生出不同适应性的语言。一门计算机语言的发展与繁荣, 与当时的社会需求, 工业界、学术界的支持推崇有着极为重要的联系。Go 语言则是满足网络中海量数据、高并发、分布式的特性而产生。

本文就一般程序设计语言的分析方法, 对 Go 语言进行了简要的分析与评价。程序设计语言最主要的作用就是形式化地描述计算过程。对此考虑两个基本特性: 效率与规律性。对于 Go 语言的应用场景, 其简单简洁性、并行性也十分关键。这就组成了本文的基本结构框架。

第 2 节简要地介绍了 Go 语言的发展简介, 主要浓缩了 Go 语言发展的重要事件与关键特性。第 3 节对 Go 语言进行了简单性分析, 有语言的基本结构特性写法直观展示与常用语言进行比较评价。第 4 节, 对 Go 语言进行规律性分析, 细分至一般性、正交性、统一性, 通过实例来对比讲述 Go 语言的改进之处。第 5 节, 则是对 Go 语言的效率分析, 主要通过三种其它常用高级语言自定义“计数程序”来比较分析 Go 语言的执行效率。第 6 节, 用了较多篇幅, 阐述 Go 语言的并行机制, 并将之前的“计数程序”并行化, 以实践验证其简单、高并发的特性, 并且对照了 Java 与 C++ 的并行机制, 比较其各自特点与优劣。最后则是自身总结感悟, 与对未完成工作的展望。

## 2 Go 语言发展简介

Go 是从 2007 年末由 Google 公司员工罗伯特格·瑞史莫 Robert Griesemer, 罗勃特·派克 Rob Pike, 肯尼斯·汤普逊 Ken Thompson 主持开发, 到 2009 年作为开源项目发布, 再到 2012 年发布第一个稳定版本 Go1.0, 现已拥有一个活跃开放的开源社区, Go 语言的形成与发展都得益于社区的大量参与协作。Go 编程语言是一个使得程序员更加有效率的开源项目。Go

具有可表达性、简洁清晰性、高效的特点。而它的新型类型系统使灵活和模块化的程序构建成为可能。Go 可以快速编译成机器代码，但也有垃圾收集的便利性和运行时反射的能力。它是一种快速、静态类型、编译的语言具备 C 语言一样的运行效率，感觉就像是动态类型的、解释的语言像 Python 一样为开发人员提供简洁的语言环境以达到相近的开发效率。

### 3 简单性分析评价

Go 语言的简单简洁性，可以从它的理念就能感觉到，“用更少的代码做更多的事”，而这里的简单性，意味着 Go 语言拥有很好的可写性与可读性，在开发效率方面有不俗的成就。

#### 3.1 变量赋值简化

Go 语言中在变量赋值方面与 C、C++ 有所不同，与 Python 比较相似，如图 1 所示：

```
var i int
var s string
i = 15          i := 15
s = "test"      等价于 s := "test"
```

图 1. Go 语言变量声明与赋值

“:=” 声明与与赋值语法糖，通过值逆向推导类型。也可以看到表示每条语句结尾的分号 “;” 被省略了，但它并不是完全不可用，当在一行中有多条语句时，可以用分号分隔，对此我们是不推崇的编码风格，所以 “;” 的省略也是其简洁性的体现。不仅如此，“:=” 还能完成平行声明并赋值多个变量，如图 2 所示，第一个式子完成了三个 int 类型的赋值与声明，第二个式子分别完成了 bool、float32、string 类型的赋值与声明：

```
a, b, c := 1, 2, 3
d, e, f := true, 1.0, "test"
```

图 2. Go 语言平行声明与赋值

当我们需要声明多个变量，并不准备不赋初始值时，可以用到关键字 var，并使用它的成组声明，常量声明 const 与包导入 import 关键字也能如此使用，如图 3 所示。对应于 C、C++、java，每当需要导入头文件或者包时或者定义常量时，往往关键字 const，import 会重复地一行一行编写，而 Go 语言简化了这些关键字的重复：

```
var(          )import (          )const(
    a int      "fmt"          d = true;
    b bool     _ "io"         e string = "test"
    s string   _ "os"         f = true
)              )              )
```

图 3. Go 语言分组声明

### 3.2 控制结构语句的简化

If 语句, 取消判断表达式的大括号, 并且能够像 C++ 语言中 for 语句一样接受局部变量初始化语句:

```
if isOk := del(); isOk == true{  
    doSome()  
}
```

switch 语句, 执行过程从上至下直到找到匹配项, 若无匹配项自动匹配 true, 表达式可以不是常量或整数, 在此 switch 语句对比 C、C++ 取消了 break 的使用(再一次简化), 即执行完一个 case 后, 不会继续运行此后面的 case, 正如图 4 左侧语句所示。它也可以用来完全替代 if-else if-else 语句, 相比 if-else if-else, 代码不再冗长, 结构更为清晰, 如图 4 右侧语句所示:

<pre>switch x {     case 0, 1, 2: f1()     case 3, 4, 5: f2()     default: f3() }</pre>	<pre>switch x := f(); {     case x &gt; 0: f1()     case x &lt; 0: f2()     case x == 0: f3() }</pre>
---	---

图 4. Go 语言 Switch 语句

### 3.3 函数多值返回, 设计简化

Go 语言对于编译语言而言, 其函数可以返回多个值, 这点与 Python 又极为相似。这样带来的好处是, 减少不必要的数据结构创建, 例如在 C 和 C++ 中, 想要返回多值, 需要定义一个 struct 或者 class, 然后在函数对该结构体或类进行操作, 最后返回此类。这看来有时候为此设计一个 struct 或者 class 是一种多余, 并且返回此类会将此类所有属性返回, 而不能按需返回。这也改进了 C 语言中的惯例, 通过修改参数值来返回, 返回值则只是用以错误标识。Go 语言的多值返回, 可以准确返回所希望返回的值。正如下函数编写, 寻找数组中第一个素数, 并返回它的值与序号, Go 语言编写将十分简单, 这就是多值的简化优点:

```
func getValueAndIndex(data [10] int) (int, int){  
    for index, value := range data{  
        if index < cap(data) && isPrime(value){  
            return index, value  
        }  
    }  
}
```


## 4 规律性分析评价

Go 简单而灵活, 它吸收了 C 语言与 Python 语言的优点, 在规律性方面, 主要体现在语言特性融为一体的良好程度, 接下来就一般性、正交性、统一性与其它语言的对照分析进行阐述, 可以看出 Go 语言相对于 C、C++ 在某些方面做出的改进。

## 4.1 一般性

Go 语言支持变长数组，和 Python 语言极为相似的，它很好地支持切片 slice，其切片与数组在声明时的区别，在于 “[ ]” 中是否填写长度，若填写长度表明是定长数组，若为空则表明是变长切片。数组方面对于 Pascal 更具有一般性。

C 语言中相等运算符 “==” 可以用来比较元素，不能直接作用与两个数组，以比较数组是否相等。而 Go 语言中直接支持数组的比较。代码 “ArrayComp.go”，详见附录。程序运行结果如下：

```
 <4 go setup calls>
array a = [1 2 3]
array b = [1 2 3]
array c = [1 1 1]
a == b = true
a == c = false

Process finished with exit code 0
```

图 5. Go 语言，数组 “==” 比较结果

结果分析：如上图 5 运行结果显示，Go 语言能够直接简单地通过 “==” 进行数组的比较。在此方面，Go 语言比 C 语言更加具有一般性。

## 4.2 统一性

我们知道 C++ 中，类的定义之后必须有分号，而函数定义之后这不需要分号，这是 C++ 中的非统一性的点。而对于 Go 语言，取消了分号的分割，并且格式上的相似，并且在 Go 语言上，将更容易写出良好统一风格的代码，统一性更好。

## 4.3 正交性

在 C 语言中，参数的传递是存在非正交性的，传递数组时为传引用，传递其它参数时按值传递。而在 Go 语言中，函数调用全部都是传值，即 pass-by-value，包括 array、slice、map、chan 等多种基本类型还有自定义类型。下面程序对比可以验证，由此可见 Go 语言中函数参数为数组时是传值的，这里摒弃了 C 语言的非正交性。

编写简单的函数，传递参数为数组类型，在函数体中，修改数组中值，在函数体外输出数组值，代码 “test.c” 与 “Test.go” 在附件中，程序运行结果：


```
 D:\学习\大三下\程序语言设计\期末论文\test.exe
1 2 3
1 0 3
```

图 6. C 语言传数组程序运行结果

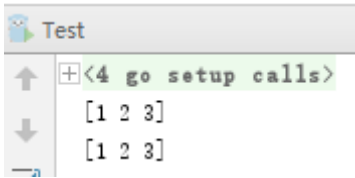


图 7. Go 语言传数组程序运行结果

由上图 5、图 6，程序运行结果可以看出，c 语言函数传递数组为传引用，修改后输出数组值已经发生改变，而对于 Go 语言，在函数运行后，数组值仍未发生变化，其数组传递是传值的，Go 比 C 在这方面更具有正交性。

5 效率

程序设计语言重要原则之一就是效率原则，它几乎涵盖了所有其他原则，而最主要的就是执行效率，翻译效率，编程效率。Go 语言在这三者上面都有不错的优化。

5.1 执行效率

这里使用“整数计数程序”来运行比较。“整数计数程序”伪代码如下：

```
sum = 0;
for(long i = 0; i < 1000000000; ++i){
    sum += 1;
}
cout << sum << duration time;
```

这是一个很简单的程序，只是对长整数进行很多次的累加运算，并观察它们的运行时间。分别编写代码“efficient.cpp”、“efficient.go”、“efficient.java”、“effice.py”四个小程序，分别用 C++、Go、JAVA、Python3 编写。其执行结果如下所示：

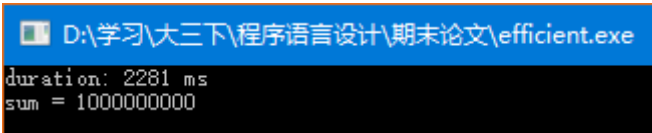


图 8. C++计数程序运行结果

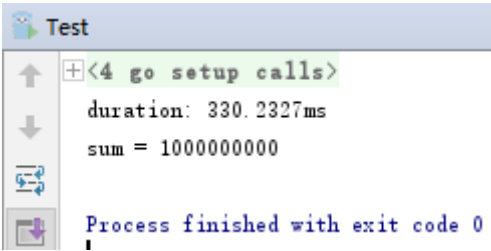


图 9. Go 语言计数程序运行结果

```
D:\学习\大三下\程序语言设计\期末论文>javac efficient.java
D:\学习\大三下\程序语言设计\期末论文>java efficient
duration: 394ms
sum = 1000000000
```

图 10. JAVA 计数程序运行结果

```
>>>
===== RESTART: D:/学习/大三下/程序语言设计/期末论文/effice.py
duration: 273.32636642456055 s
sum = 1000000000
```

图 11. Python3 计数程序运行结果

语言	执行时间	结果正确性
Go	330ms	正确
JAVA	394ms	正确
C++	2281ms	正确
Python3	273s	正确

表格 1. 各语言计数程序运行结果

结果分析：Go 语言是编译类语言，它的执行效率很高，与 C++ 在同一个级别上，还有 JAVA 是解释型语言，它的执行效率也同样非常之高。值得注意的是，使用 Python 动态解释型的强类型定义语言时，可以看到它的执行时间远大于另外三者。由于这里只做出整数加法的测试，执行效率还要看浮点运算能力，字符串处理、递归调用等能力，这里并不能直接判定 C++、Go、JAVA 的执行效率孰好孰坏，但是可以知道它们的效率是要比 python 好得多。综上 Go 语言编写出来的程序，在可写性与可读性上来看，它和 python 一样干净简洁，化繁为简，而它的执行效率却比它们高效的多，拥有着编译型语言普遍的运行效率，这是难能可贵的，吸收了 C 与 Python 的各自优点。

## 5.2 翻译效率、编码效率

Go 语言是编译型语言，利用 go build 命令，能够快速地进行源代码的翻译、连接、生成可执行文件过程。Go 拥有快速的编译时间，这里并没有找到一个可以显式输出编译时长的方法，但从感官上 go build 时间是十分迅速的。有个有趣的事，当时 Go 在 Google 准备立项前，其设计者正是饱受 C++、JAVA 编译慢的折磨。

对于编码效率，在之前的段落中已经可以看到，Go 语言与 Python 语言的形式上有许多相似处，并且在附件所有的代码中也可以有所感受，编写 Go 语言是极简的，甚至可以用“懒”来形容。但这很大程度上简化了编码者的工作，更少的字符做更多的工作，在工业界，这种简单易写显得十分重要。这也凸显了 Go 语言的开发效率是值得推崇的。

## 6 并行性对比分析与评价

### 6.1 并行处理简介

根据摩尔定律, 在单个 CPU 核心上的性能提升, 终将进入一个平缓期, 效果将再够显著。而随着半导体技术发展趋于物理极限, 单核心早已不能满足日益增长的高效率高性能需求, 不论在 PC 端还是移动端, 四核八核早已是各大厂商争相宣传的配备。多核时代, 并行程序设计技术备受瞩目。而在 Go 语言在刚开始设计之时, 就强调了高并发的机制, 并发性是 Go 语言一大特点之一, 使得自身能够高效利用多核与互联的计算机网络环境, 面对海量数据需求, 或者游戏等高并行应用时都有更为轻量更为有效的解决方案。

什么是并行? 并行指多个处理器或多核处理器同时处理多个不同的任务, 在物理上同时发生。对于并行性, 首要解决的是进程管理、进程通信两个问题。四种常见的并行程序设计模型: 共享内存模型、消息传递模型、数据并行模型、面向对象模型。

Go 语言并发机制基于的是消息传递模型, 消息传递模型是指程序中不同进程之间通过显式方法 (如函数调用、运算符等) 传递消息实现显式通信, 实现进程之间的数据交换、同步控制等。消息包括指令、数据、同步信号等。



图 12. Go 语言并行机制-消息传递模型

### 6.2 Go语言并行编程与实例

正如图 11 所示, Go 语言中 goroutine/channel 机制就是用来实现并行编程的需求, 有效地支持高并发特性。goroutine 是一种轻量化的线程, 简单并且消耗少, 它与其它 goroutine 拥有相同地址空间的函数并行执行。保留字 go 作为开头, 则表明完成一个 goroutine 的生成启动。而对于 Goroutine 之间的通信机制则需要 channel 来实现, channels 的想法来自于 CSP 通信顺序进程模型。channel 就像一个自定义的管道, 连接 goroutine, 用来收发数据。值得注意的是, 通过 channel, 任何 goroutine 的读写操作都是阻塞的, 这样就使得 Goroutines 同步变的更加的简单, 而不需要显式的加锁, 使用 Select 函数, 能够监听 channel 上的数据, 可以记录其相应数量, 避免所有进程都睡眠而导致的死锁情况。但注意, goroutine 此时可能只是并发的, 而非并行计算的, 可以使用 runtime.GOMAXPROCS(n) 来设置 goroutine 并行执行个数, 即当前使用运行 CPU 核心数。

使用 goroutine/channel 机制重写“计数程序”, 并使用不同的 goroutine 并行数量。代码“ParallelCal.go”, 详见附录。程序运行结果如下:



图 13. Go 语言并行计数程序结果（不同核心数）

结果分析：可以发现，当核心数唯一时，它和之前所运行的串行计算情况的时间消耗差不多，当和核心数增长时，其时间消耗降低，这是并行计算所带来的性能提升，注意到当设置核心数 $\geq 4$ 时，其时间消耗基本不再减少，而是稳定波动，这表明本机只有 4 核处理性能。从运行时间看，很显然，并行计算时耗时更少并随着并行进程数量线性减少，但是其减少并不夸张显著，可以认为在此使用并行计算时还是需要一些额外消耗，当程序复杂度攀升，这些消耗可以忽略时，并行计算所带来的效率提高将更加显著。

### 6.3 JAVA与C++并行机制

在 Java 语言中支持多种并发模型，在这里，我们讨论其中的 ForkJoin 模型，此模型框架采纳了“分治算法”的思想，把大任务逐步细分为小任务，通过并行计算小任务所得结果汇总成最后大任务，完成目标。这很类似于四种常见的并行程序设计模型中的数据并行模型，如下图所示。



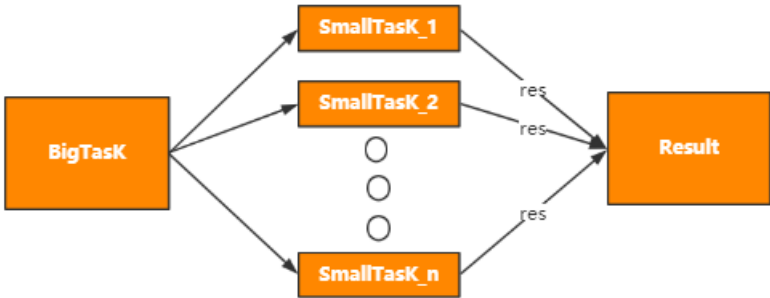


图 14. JAVA 中 ForkJoin 并发框架

使用 JAVA 语言中 ForkJoin 机制重写“计数程序”。代码“ParalleCal.java”，详见附录。程序运行结果如下图 13，表明 Java 语言中并行计算也带来了性能的稳定提升

```
D:\学习\大三下\程序语言设计\Test>javac CountTask.java
D:\学习\大三下\程序语言设计\Test>java CountTask
Parallel time with 4 processors is 212 milliseconds
sum=1000000000
```

图 15. JAVA 并行计数程序结果

在 C++中，则采用的是共享内存模型，即利用共享变量来实现隐式进程间的通信，而这里关键的问题则是数据一致性与同步，最为常用的方法就是通过锁机制来实现。不同线程都具有访问共享内存的权限，如果对共享内存进行读写，需要必须考虑到互斥的问题，避免产生计算冲突。“计数程序”用 C++实现。代码“ParalleCal.cpp”，详见附录，其执行效果如下：

```
G:\Program Files\Visual Studio 2015\Projects\Project1\..
duration: 250 ms
sum = 1000000000
请按任意键继续...
```

图 16. C++并行计数程序结果

结果分析：C++中用共享内存模型进行并行计算，效果依旧十分有效，极大地提升了运行效率。

6.4 Go、JAVA、C++并行机制评价与总结

在经过上诉对三种语言并行编程后，分析其执行结果，与代码（附录中可见）的复杂程度，还有其模型原理概念的实践了解，总结表格 2 如下：

语言	串行计算时间	并行计算时间	通信机制	同步机制	可读可写性
Go	330ms	155ms	消息传递模型，goroutine/channel 机制，发送消息，显示进行通信	消息的发送必须在消息的接之前，同步隐式进	代码简单易懂，无需考虑互斥锁。可读性可写性好。

				行。	
JAVA	390ms	212ms	数据并行模型，ForkJoin 并行框架。“分治算法”思想实现。	在小任务中计算，在主任务中汇总。	对任务进行面向对象处理，需要懂得如何构建，可读性可写性一般（代码量最大）
C++	2280ms	250ms	共享内存模型，线程之间共享公共变量，对此写读来隐式进行通信。	同步显式进行。通过对共享变量加锁处理实现。	互斥同步策略逻辑需要确认，代码复杂，可写性较差

表格 2. Go、JAVA、C++并行机制总结

7 总结与展望

7.1 总结

Go 语言的简洁性是最突出的特点，“少既是多”，不仅仅是形式语法上，在程序设计上与并行机制结构上都有很直观的体现，这也是它被工业界所极力推崇的原因。最基本的，Go 在声明赋值语句、控制结构语句、函数多值返回中都有所简化。规律性方面，参数传递都是传值调用，这相对于 C 语言是正交性方面的优化。Go 支持变长数组与切片功能，“==”符号支持数组比较，比 C 语言更具有一般性。效率方面，Go 语言在整数运算方面依旧有不俗表现，对比于代码风格与形式相似的解释语言 Python，它的执行速度高了一个数量级甚至某些时候比 java 与 C++表现更为良好。最后并行性，Go 语言十分简练地使用消息传递模型，Channel/Goroutine 机制配合，使得编写高并发程序极为顺畅，逻辑简单，可写性强，这是 C++与 JAVA 所不具备的，但也是因为消息传递方式，效率优化相较于 C++的共享内存模型较低，并行性受到影响。

总的来说，Go 语言做到了解释性语言（如 Python）一般的简单性，还有着编译型语言（如 C）一般的高效率。

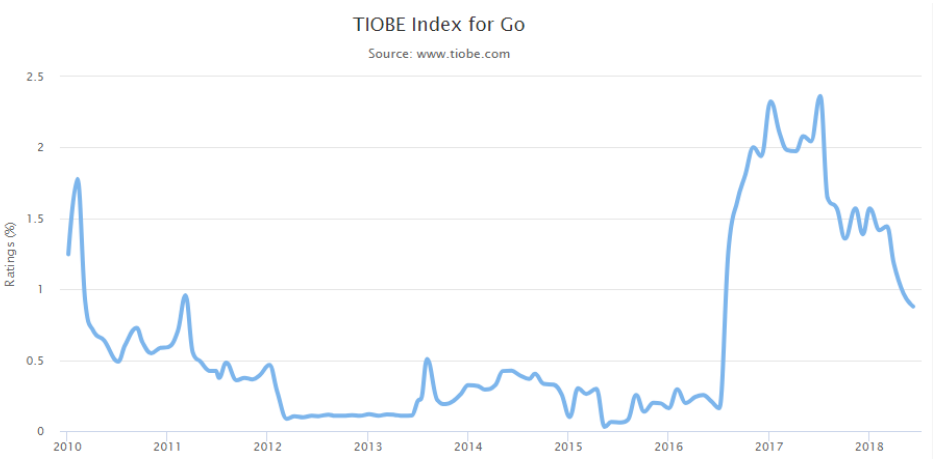


图 17. Go 语言活动趋势--来源于 TIOBE

## 7.2 展望

由于时间有限, 最终的成果就是上述这些分析评价结果。这里记录一下待解决的问题, 以便开展后续的工作。第一, Go 语言还有许多其它特性与一般常用语言不同, 如垃圾处理机制、动态式函数语言支持、面向对象编程等。第二, Go 语言的不足之处分析, 本文中几乎都在分析 Go 语言的优势之处, 而对 Go 语言的不足之处所谈甚少, 在图 17 中是 Go 语言的趋势活动, 从 2017 年末尾, Go 语言热度一直下降, 而对它的缺点分析将有助于了解其中缘由, 这也是一个较大的课题。第三, Go 语言编译效率探究, 在效率一节中, 对于编译效率, 并没有给出一个定量的结论, 从其发展原由与代码实践来看, 由于工程量不算大, 也并没有找到一个可以输出编译时间的工具, 这是需要改进之处 (如果读者有什么发现与想法可与我联系)。

## 8 感想

对于未来程序设计语言的发展, 我认为仍会秉持极简主义的理念, “少既是多”, 也将不断地向高层抽象。编码上不断地做“减法”, 一些机械繁杂的工作就让计算机去完成, 而让编码者更加解放, 以专注于“想要做什么”的问题上去。

每一种语言都自身独特的性质, 我们不能以业余的眼光去看待一个语言, 也不需执着于活跃的热门语言, 语言的热门程度只有当前社会需求极大地相关, 而与语言本身的优劣好坏没有本质联系。通过 2018 年春季学期的程序设计语言设计原理与实践课程, 最大的收获不是掌握或者熟悉某一中热门语言, 而是学会一种分析评价的方法与方式, 通过实验验证, 加深该语言某个特征属性的印象, 并以之有更准确量化的评价指标。我自己在课程中, 与课后花费也许久的精力与时间按, 也获得了很大的收获。

感谢: 最后, 非常感谢, 张老师一学期的辛苦教学。

## 9 参考文献:

- [1] 邓楠. Ready? Go!语言开发背景、语法和类型.《程序员》期刊. 2012 年 第 5 期:122-125.
- [2] 邓楠,七叶,陈佳桦,谢孟军. Go 语言技巧.《程序员》期刊. 2014 年 第 3 期:51-55.
- [3] 史建国,黄春,王戟.X10 与 Go 语言的并行机制比较研究[C].2011 全国软件与应用学术会议 (NASAC2011)论文集.国防科学技术大学,2011:175-179.
- [4] 林荣智.GO 语言的并发编程介绍[J].科技展望,2016,(22):12-12.DOI:10.3969/j.issn.1672-8289.2016.22.009.
- [5] Miek Gieben. LEARNING GO. 2010-2012.  
<http://85g.net/document/Go%E8%AF%AD%E8%A8%80%E5%AD%A6%E4%B9%A0.pdf>
- [6] Meyerson Jeff, "The Go Programming Language". Software, IEEE, Vol. 31, No. 5, pp. 104-104, Mar. 2014.
- [7] Rob Pike. "The Go Programming Language". In: Talk given at Google's Tech Talks (2009).
- [8] Loudon, K.C.: Programming Languages: Principles and Practice. Second Edition PWS-Kent Publishing Co., Boston (2002)
- [9] java 并行计算 Fork 和 Join 的使用[EB/OL].  
<https://www.cnblogs.com/rwxwsblog/p/6231542.html>.
- [10] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber. Mathematizing C++ concurrency. In Proc. POPL, 2011.
- [11] Go 语言中文网[EB/OL].  
<https://studygolang.com/>

## 10 附录

### 1. “ArrayComp.go” 支持 “==” 作用与数组，一般性验证

```
package main

import "fmt"

func main() {

    a := [3] int{1,2,3}
    b := [3] int{1,2,3}
    c := [3] int{1,1,1}

    fmt.Println("array a =", a)
    fmt.Println("array b =", b)
    fmt.Println("array c =", c)
    fmt.Println("a == b =", a == b)
    fmt.Println("a == c =", a == c)
}
```

### 2. “test.cpp” C 语言传数组为参数，非正交性

```
#include<stdio.h>

void modifyArray(int a[3])
{
    a[1] = 0;
}

int main()
{
    int a[3] = {1,2,3};

    int i;
    for(i = 0; i < 3; ++i)
        printf("%d ",a[i]);
    printf("\n");

    modifyArray(a);

    for(i = 0; i < 3; ++i)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

```
}
```

### 3. “Test.go” Go 语言传数组

```
package main
import (
    "fmt"
)

func main() {
    a := [3]int{1,2,3}
    fmt.Println(a)
    modifyArray(a)
    fmt.Println(a)
}

func modifyArray(data [3]int) {
    data[1] = 0
}
```

### 4. “efficient.cpp” C++计数程序执行效率

```
#include<iostream>
#include<windows.h>
using namespace std;

int main()
{
    long sum = 0;

    DWORD startTime = GetTickCount();

    for(long i = 0; i < 1000000000; ++i){
        sum += 1;
    }

    DWORD endTime = GetTickCount();

    cout<<"duration: "<<double(endTime - startTime)<<" ms"<<endl;
    cout<<"sum = "<<sum<<endl;

    return 0;
}
```

### 5. “efficient.go” Go 计数程序执行效率

```
package main
```

```
import "fmt"
import "time"

func main() {
    now := time.Now()
    var sum int64 = 0
    var i int64 = 0
    for ; i < 1000000000; i++ {
        sum += 1
    }
    duration := time.Since(now)
    fmt.Println("duration:", duration)
    fmt.Println("sum =", sum)
}
```

#### 6. “efficient.java” JAVA 计数程序执行效率

```
public class efficient {
    public static void main(String []args){
        long sum = 0;

        long startTime=System.currentTimeMillis(); //开始时间
        for(long i = 0; i < 1000000000; ++i){
            sum += 1;
        }

        long endTime=System.currentTimeMillis(); //结束时间
        System.out.println("duration "+(endTime-startTime)+"ms");
        System.out.println("sum = "+sum);
    }
}
```

#### 7. “effice.py” Python3 计数程序执行效率

```
import time
start = time.time()

sum = 0
i = 0
while i < 1000000000:
    sum += 1
    i += 1
```

```
end = time.time()

print('duration: ', end - start, 's')
print('sum = ', sum)
```

## 8. “ParalleCal.go” Go 语言并行计数程序

```
package main

import (
    "fmt"
    "time"
    "runtime"
)

func getSubSum(start int, end int, resultChan chan int){
    sum := 0
    for i:=start; i < end; i++){
        sum += 1;
    }
    resultChan <- sum
}

func main() {
    runtime.GOMAXPROCS(4)

    resultChan := make(chan int, 3)
    const STEP = 10000000
    const N = 100000000
    t_start := time.Now()

    for i := 0; i < N; {
        go getSubSum(i, i + STEP, resultChan)
        i += STEP
    }

    sum := 0
    for i:=0; i < 100;{
        select {
        case subSum := <-resultChan:
            sum += subSum
            i++
        }
    }
```

```

    }

    duration := time.Since(t_start)
    fmt.Println("核心数 =", 4)
    fmt.Println("duration:", duration)
    fmt.Println("sum =", sum)
}

```

## 9. “ParalleCal.java” JAVA 语言并行计数程序

```

import java.util.ArrayList;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;

public class CountTask extends RecursiveTask<Long> {
    private static final int THRESHOLD = 10000000;
    private long start;
    private long end;

    public CountTask(long start, long end) {
        this.start = start;
        this.end = end;
    }

    @Override
    protected Long compute() {
        long sum = 0;
        boolean canCompute = (end - start) <= THRESHOLD;
        if (canCompute) {
            for (long i = start; i < end; i++) {
                sum += 1;
            }
        }
        else {
            long step = THRESHOLD;
            ArrayList<CountTask> countTasks = new ArrayList<>();
            long pos = start;
            for (int i = 0; i < 100; i++) {
                long lastOne = pos + step;
                if (lastOne >= end) {

```



```

        lastOne = end;
    }
    CountTask countTask = new CountTask(pos, lastOne);
    pos += step;
    countTasks.add(countTask);
    countTask.fork();
}
for (CountTask task : countTasks) {
    sum += task.join();
}
}
return sum;
}

public static void main(String[] args) {
    long startTime;
    long endTime;
    startTime = System.currentTimeMillis();
    ForkJoinPool forkJoinPool = new ForkJoinPool();
    CountTask task = new CountTask(0, 1000000000L);
    ForkJoinTask<Long> result = forkJoinPool.submit(task);
    try {
        long res = result.get();
        endTime = System.currentTimeMillis();
        System.out.println("\nParallel time with
"+Runtime.getRuntime().availableProcessors()+" processors is
"+(endTime-startTime)+" milliseconds");
        System.out.print("sum=" + res);
    }
    catch (InterruptedException e) {
        e.printStackTrace();
    }
    catch (ExecutionException e) {
        e.printStackTrace();
    }
}
}

```

## 10. “ParalleCal.cpp” C++语言并行计数程序

```

#include <iostream>
#include <thread>

```

```
#include <mutex>
#include<Windows.h>
using namespace std;

mutex sumLock;
long sum = 0;

void subSum(long start, long end)
{
    long subsum = 0;
    for (long i = start; i < end; ++i)
    {
        subsum += 1;
    }
    sumLock.lock();
    sum += subsum;
    sumLock.unlock();
}

int main()
{
    const long STEP = 10000000;
    const long N = 1000000000;

    DWORD startTime = GetTickCount();

    for (long i = 0; i < N;)
    {
        thread* th;
        th = new thread(subSum, i, i + STEP);
        th->join();
        i += STEP;
    }

    DWORD endTime = GetTickCount();

    cout << "duration: " << double(endTime - startTime) << " ms" << endl;
    cout << "sum = " << sum << endl;

    system("pause");
    return 0;
}
```