第九章：Yak虚拟机 - YakVM

9.1 背景知识

在学习完 Yak 的基本使用后，学有余力的读者可以试着了解Yak虚拟机即YakVM。要注意，本章节是独立而且完全可选的，读者可以选择性学习。

虚拟机（Virtual Machine，简称vm）简单地来说是一台虚拟机器，我们可以在这台机器上执行指令。值得注意的是，这些指令并不是通常所理解的代码，而是经过转换后的字节码（Bytecode）。根据不同的字节码，虚拟机会执行不同的指令，最终实现复杂的编程逻辑。不同类型的字节码则组成了指令集（Instruction Set Architecture，简称ISA）。

YakVM是一个基于Golang语言的虚拟机，它运行方式基于栈的操作，在栈虚拟机中，大部分指令都是以栈为操作对象的。我们熟知的语言中，java的虚拟机JVM就是一种栈虚拟机。

9.2 栈结构

在简单了解了什么是 YakVM 之后，我们还需要了解一下栈结构。

栈（Stack）是一种特殊的数据结构，它的特殊之处在于它只允许在一端（我们称之为"顶部"）添加或者删除数据。这种方式被称为后进先出（Last In First Out，简称LIFO）。这意味着最后放入栈中的元素总是第一个被取出。

读者可以把栈想象成一个堆叠的餐盘。在一家自助餐厅里，餐盘通常会堆叠在一起，当需要一个餐盘时，会从最上面的一堆中取一个，而不是从中间或底部取。同样，当要放回餐盘时，会把它放在最上面。这就是栈的工作方式。

在栈中，主要有两个主要的操作：入栈（Push）和出栈（Pop）。入栈就像在餐盘堆上放一个新的餐盘，而出栈就像从堆上取走一个餐盘。

还有一个操作叫做“查看栈顶”（Peek），这就像在查看餐盘堆顶的餐盘是什么样子，但是并不取走它。

9.3 指令集与实际应用

前面提到，指令集是不同类型字节码的集合。在YAKVM中一共存在81种字节码，它们一起组成了 YAKVM 的指令集。

为了让读者更好地理解 YAKVM 的指令集，我们以一段简单的 Yak 代码起步，以此来学习 YAKVM 的指令集。

一段简单的加法代码如下：

```
a = int(input())
b = a + 1
println(b)
```

这段代码接收一个数字的输入，将其强制转换为整数后赋值给 b ，然后再加一，赋值给a后使用 println 进行输出。

由于完整的字节码可能比较长，我们按行分析。其对应的第一行字节码如下：

```
1:8->1:12       0:OP:pushid             input
1:13->1:14      1:OP:call               vlen:0
1:4->1:6        2:OP:type               int
1:4->1:15       3:OP:type-cast
1:4->1:15       4:OP:list               1
1:0->1:0        5:OP:pushleftr          1
1:0->1:0        6:OP:list               1
1:0->1:15       7:OP:assign
```

第一个字节码为 pushid ，它将 input 这个id放入栈上，紧接着第二个字节码为call，它从栈上取出一个值作为函数名，并通过这个值找到对应的函数进行调用，并将调用的返回值放入栈上。在这里对应的是调用 input 这个函数。

第三个字节码将int类型放入栈上，随后第四个字节码从栈上取出两个值，第一个值为 type ，第二个值为要转换的类型，在这里对应的是将 input 这个函数的返回值强制转换为int类型。

随后的四个字节码分别是：取出栈上的1个值并转换为 list ，从栈上放入一个左值，然后再取出栈上的1个值并转换为list，最后从栈上取出2个值进行赋值操作，这里对应的是赋值操作，即将强制转换类型后的值赋给a这个变量。

读到这里，读者可能会疑惑 list 这个字节码的作用，实际上这是为了支持多赋值的情况，即支持形如: a, b = 1, 2 这样的赋值。

接下来我们再看看第二行的字节码，很多与第一行字节码相似：

```
2:4->2:4         8:OP:pushr             1
2:8->2:8         9:OP:push              1
2:4->2:8        10:OP:add
2:4->2:8        11:OP:list              1
2:0->2:0        12:OP:pushleftr         2
2:0->2:0        13:OP:list              1
2:0->2:8        14:OP:assign
```

第一个字节码为 pushr ，这个字节码将对应的数字1指向的变量放入栈上，在这里指向的是变量 a 。随后第二个字节码将数字1放入栈上，第三个字节码从栈上取出两个值并相加，将结果再放入栈上。

紧随其后的字节码就与上述第一行的字节码相似了，它们做的事情就是将相加的结果赋值给变量 b 。

最后来看看第三行的字节码：

```
1 3:0->3:6        15:OP:pushid              println
2 3:8->3:8        16:OP:pushr               2
3 3:7->3:9        17:OP:call                vlen:1
4 3:0->3:9        18:OP:pop
```

第一个字节码将 println 这个 id 放入栈上，第二个字节码将2指向的变量放入栈上，在这里指向的是变量b。紧接着第二个字节码为 call ，需要注意的是这里 vlen 等于1，所以它先从栈上取出1个值作为参数，再从栈上取出一个值作为函数名，并通过这个值找到对应的函数进行调用，并将调用的返回值放入栈上。这里实际上对应的代码就是 println(b) 。最后一个字节码是 pop ，它从栈上取出一个值，这个字节码保证了最终的栈平衡，即所有字节码执行完毕后栈的长度为0。

根据上述例子，实际上不难理解 YAKVM 的字节码的作用。编译器遍历代码，生成语法树，再根据语法树生成出比代码颗粒度更小的字节码，最后执行，从而实现了复杂的代码逻辑。

读者如果对 YAKVM 的字节码感兴趣，想要自己尝试将编写的代码转换为字节码的话，可以尝试使用 Yak 自带的 cdebug 模式。 cdebug 模式允许用户在命令行下通过交互式的方法调试Yak代码，在这个模式下，我们也可以通过执行 sao 这个命令来获取并查看代码编译出来的所有字节码。具体操作如下：

9.4 Goroutine管理

Yaklang支持类似于Golang的go关键字，用于创建goroutine。在YakVM中，每个goroutine都是平级的，拥有独立的虚拟机栈，它们之间没有父子关系。

当使用go关键字调用函数时，会创建一个新的虚拟机栈，并将当前函数的栈帧推入栈中。在一个虚拟机栈内，当最后一个栈帧被弹出（POP）后，虚拟机栈将被销毁。

特别值得注意的是，由于闭包函数的支持，栈帧内会储存函数定义时的作用域和父作用域。因此，可能会出现多个goroutine调用同一块作用域的情况。不同goroutine对于同一块作用域的同一个变量的访问可能存在风险。Yaklang提供了并发安全的变量类型channel，当一个goroutine调用时，会阻塞其他goroutine的访问。

在一些场景中，需要保证goroutine的有序执行，这时可以使用channel来进行顺序控制。

Yaklang不允许主动销毁goroutine，但可以通过channel向其他goroutine发送结束信号。当goroutine读取到结束信号后，自主判断是否结束。当主线程结束后，所有goroutine会被强制销毁。

9.5 Golang标准库复用

Yaklang的这一功能的实现原理涉及到YakVM创建时定义的一组Golang标准库到Yaklang函数名的映射。这个映射充当了一个桥梁，将Golang标准库与Yaklang集成在一起。当Yaklang代码执行时，通过这个映射，YakVM可以找到相应的Golang标准库函数，实现了Golang函数的直接调用。

在执行过程中，YakVM将根据Golang函数的参数类型自动将Yaklang类型变量按照一定规则转换为Golang的变量类型。这个过程是关键的，因为它确保了参数的正确匹配，从而保证了函数调用的准确性。一旦参数准备就绪，YakVM将调用Golang函数，并将函数执行结果转换为Yaklang类型，以便传递给YakVM的其他部分。

这种实现原理的优势在于，它使Yaklang能够充分利用Golang的庞大生态系统，将Golang的性能和功能扩展到Yaklang编程环境中。这为开发者提供了更多的可能性，可以使用Golang的强大特性，同时仍然享受Yaklang的简洁和灵活性。

9.6 多语言兼容

YakVM作为一种虚拟机，在其字节码设计上具有出色的灵活性和可扩展性，这使得它成为一个强大的语言后端。这个设计之所以如此重要，是因为它允许YakVM支持多种前端语言，实现了对不同编程语言的广泛兼容性。

YakVM的字节码设计是其强大功能的核心之一。字节码是一种中间代码，它是由编译器生成的，可以在虚拟机中执行。YakVM的字节码是精心设计的，以在执行过程中提供高效的性能和灵活性。这些字节码指令包括了各种操作，如变量赋值、函数调用、控制流操作等等，使得YakVM可以支持各种高级语言的编译输出。

更重要的是，YakVM的字节码设计是可扩展的。这意味着新的指令和功能可以相对容易地添加到虚拟机中，而不需要对YakVM的核心进行大规模修改。这种可扩展性使得YakVM能够不断适应不同语言的需求，同时为未来的语言扩展提供了坚实的基础。现已基于YakVM实现了Yaklang、NASL和Lua的前端，这意味着开发者可以使用这些语言中的任何一个，将其编译成YakVM的字节码，然后在YakVM上执行。在安全领域的显著用途是用户可以选择使用它们中的任意一种语言脚本用于安全检测。