# 跟vczh看实例学编译原理——零：序言

在《如何设计一门语言》里面，我讲了一些语言方面的东西，还有痛快的喷了一些XX粉什么的。不过单纯讲这个也是很无聊的，所以我开了这个《跟vczh看实例学编译原理》系列，意在科普一些编译原理的知识，尽量让大家可以在创造语言之后，自己写一个原型。在这里我拿我创造的一门很有趣的语言 <https://github.com/vczh/tinymoe/> 作为实例。

商业编译器对功能和质量的要求都是很高的，里面大量的东西其实都跟编译原理没关系。一个典型的编译原理的原型有什么特征呢？

1. 性能低
2. 错误信息难看
3. 没有检查所有情况就生成代码
4. 优化做得烂
5. 几乎没有编译选项

等等。Tinymoe就满足了上面的5种情况，因为我的目标也只是想做一个原型，向大家介绍编译原理的基础知识。当然，我对语法的设计还是尽量靠近工业质量的，只是实现没有花太多心思。

为什么我要用Tinymoe来作为实例呢？因为Tinymoe是少有的一种用起来简单，而且库可以有多复杂写多复杂的语言，就跟C++一样。C++11额标准库在一起用简直是愉快啊，Tinymoe的代码也是这么写的。但是这并不妨碍你可以在写C++库的时候发挥你的想象力。Tinymoe也是一样的。为什么呢，我来举个例子。

## Hello, world!

Tinymoe的hello world程序是很简单的：

module hello world

using standard library

sentence print (message)

redirect to “printf”

end

phrase main

print “Hello, world!”

end

module指的是模块的名字，普通的程序也是一个模块。using指的是你要引用的模块——standard library就是Tinymoe的STL了——当然这个程序并没有用到任何standard library的东西。说到这里大家可能意识到了，Tinymoe的名字可以是不定长的token组成的！没错，后面大家会慢慢意识到这种做法有多么的强大。

后面是print函数和main函数。Tinymoe是严格区分语句和表达式的，只有sentence和block开头的函数才能作为语句，而且同时只有phrase开头的函数才能作为表达式。所以下面的程序是不合法的：

phrase main

(print “Hello, world!”) + 1

end

原因就是，print是sentence，不能作为表达式使用，因此他不能被+1。

Tinymoe的函数参数都被写在括号里面，一个参数需要一个括号。到了这里大家可能会觉得很奇怪，不过很快就会有解答了。为什么要这么做，下一个例子就会告诉我们。

print函数用的redirect to是Tinymoe声明FFI（Foreign Function Interface）的方法，也就是说，当你运行了print，他就会去host里面找一个叫做printf的函数来运行。不过大家不要误会，Tinymoe并没有被设计成可以直接调用C函数，所以这个名字其实是随便写的，只要host提供了一个叫做printf的函数完成printf该做的事情就行了。main函数就不用解释了，很直白。

## 1加到100等于5050

这个例子可以在Tinymoe的主页（<https://github.com/vczh/tinymoe/>）上面看到：

module hello world

using standard library

sentence print (message)

redirect to "printf"

end

phrase **sum from (start) to (end)**

set the result to 0

repeat with the current number from start to end

add the current number to the result

end

end

phrase main

print "1+ ... +100 = " & **sum from 1 to 100**

end

为什么名字可以是多个token？为什么每一个参数都要一个括号？看加粗的部分就知道了！正是因为Tinymoe想让每一行代码都可以被念出来，所以才这么设计的。当然，大家肯定都知道怎么算start + (start+1) + … + (end-1) + end了，所以应该很容易就可以看懂这个函数里面的代码具体是什么意思。

在这里可以稍微多做一下解释。the result是一个预定义的变量，代表函数的返回值。只要你往the result里面写东西，只要函数一结束，他就变成函数的返回值了。Tinymoe的括号没有什么特殊意思，就是改变优先级，所以那一句循环则可以通过添加括号的方法写成这样：

repeat with (the current number) from (start) to (end)

大家可能会想，repeat with是不是关键字？当然不是！repeat with是standard library里面定义的一个block函数。大家知道block函数的意思了吧，就是这个函数可以带一个block。block有一些特性可以让你写出类似try-catch那样的几个block连在一起的大block，特别适合写库。

到了这里大家心中可能会有疑问，循环为什么可以做成库呢？还有更加令人震惊的是，break和continue也不是关键字，是sentence！因为repeat with是有代码的：

category

start REPEAT

closable

block (sentence deal with (item)) **repeat with** (argument item) **from** (lower bound) **to** (upper bound)

set the current number to lower bound

repeat while the current number <= upper bound

deal with the current number

add 1 to the current number

end

end

前面的category是用来定义一些block的顺序和包围结构什么的。repeat with是属于REPEAT的，而break和continue声明了自己只能直接或者间接方在REPEAT里面，因此如果你在一个没有循环的地方调用break或者continue，编译器就会报错了。这是一个花边功能，用来防止手误的。

大家可能会注意到一个新东西：(argument item)。argument的意思指的是，后面的item是block里面的代码的一个参数，对于repeat with函数本身他不是一个参数。这就通过一个很自然的方法给block添加参数了。如果你用ruby的话就得写成这个悲催的样子：

repeat\_with(1, 10) do |item|

xxxx

end

而用C++写起来就更悲催了：

repeat\_with(1, 10, [](int item)

{

xxxx

});

block的第一个参数sentence deal with (item)就是一个引用了block中间的代码的委托。所以你会看到代码里面会调用它。

好了，那repeat while总是关键字了吧——不是！后面大家还会知道，就连

if xxx

yyy

else if zzz

www

else if aaa

bbb

else

ccc

end

也只是你调用了if、else if和else的一系列函数然后让他们串起来而已。

那Tinymoe到底提供了什么基础设施呢？其实只有select-case和递归。用这两个东西，加上内置的数组，就图灵完备了。图灵完备就是这么容易啊。

## 多重分派（Multiple Dispatch）

讲到这里，我不得不说，Tinymoe也可以写类，也可以继承，不过他跟传统的语言不一样的，类是没有构造函数、析构函数和其他成员函数的。Tinymoe所有的函数都是全局函数，但是你可以使用多重分派来“挑选”类型。这就需要第三个例子了（也可以在主页上找到）：

module geometry

using standard library

phrase square root of (number)

redirect to "Sqrt"

end

sentence print (message)

redirect to "Print"

end

type rectangle

width

height

end

type triangle

a

b

c

end

type circle

radius

end

phrase area of (shape)

raise "This is not a shape."

end

phrase area of (shape : rectangle)

set the result to field width of shape \* field height of shape

end

phrase area of (shape : triangle)

set a to field a of shape

set b to field b of shape

set c to field c of shape

set p to (a + b + c) / 2

set the result to square root of (p \* (p - a) \* (p - b) \* (p - c))

end

phrase area of (shape : circle)

set r to field radius of shape

set the result to r \* r \* 3.14

end

phrase (a) and (b) are the same shape

set the result to false

end

phrase (a : rectangle) and (b : rectangle) are the same shape

set the result to true

end

phrase (a : triangle) and (b : triangle) are the same shape

set the result to true

end

phrase (a : circle) and (b : circle) are the same shape

set the result to true

end

phrase main

set shape one to new triangle of (2, 3, 4)

set shape two to new rectangle of (1, 2)

if shape one and shape two are the same shape

print "This world is mad!"

else

print "Triangle and rectangle are not the same shape!"

end

end

这个例子稍微长了一点点，不过大家可以很清楚的看到我是如何定义一个类型、创建他们和访问成员变量的。area of函数可以计算一个平面几何图形的面积，而且会根据你传给他的不同的几何图形而使用不同的公式。当所有的类型判断都失败的时候，就会掉进那个没有任何类型声明的函数，从而引发一场。嗯，其实try/catch/finally/raise都是函数来的——Tinymoe对控制流的控制就是如此强大，啊哈哈哈哈。就连return都可以自己做，所以Tinymoe也不提供预定义的return。

那phrase (a) and (b) are the same shape怎么办呢？没问题，Tinymoe可以同时指定多个参数的类型。而且Tinymoe的实现具有跟C++虚函数一样的性质——无论你有多少个参数标记了类型，我都可以O(n)跳转到一个你需要的函数。这里的n指的是标记了类型的参数的个数，而不是函数实例的个数，所以跟C++的情况是一样的——因为this只能有一个，所以就是O(1)。至于Tinymoe到底是怎么实现的，只需要看《如何设计一门语言》第五篇（<http://www.cppblog.com/vczh/archive/2013/05/25/200580.html>）就有答案了。

## Continuation Passing Style

为什么Tinymoe的控制流都可以自己做呢？因为Tinymoe的函数都是写成了CPS这种风格的。其实CPS大家都很熟悉，当你用jquery做动画，用node.js做IO的时候，那些嵌套的一个一个的lambda表达式，就有点CPS的味道。不过在这里我们并没有看到嵌套的lambda，这是因为Tinymoe提供的语法，让Tinymoe的编译器可以把同一个层次的代码，转成嵌套的lambda那样的代码。这个过程就叫CPS变换。Tinymoe虽然用了很多函数式编程的手段，但是他并不是一门函数是语言，只是一门普通的过程式语言。但是这跟C语言不一样，因为它连C#的yield return都可以写成函数！这个例子就更长了，大家可以到Tinymoe的主页上看。我这里只贴一小段代码：

module enumerable

using standard library

symbol yielding return

symbol yielding break

type enumerable collection

body

end

type collection enumerator

current yielding result

body

continuation

end

略（这里实现了跟enumerable相关的函数，包括yield return）

block (sentence deal with (item)) repeat with (argument item) in (items : enumerable collection)

set enumerator to new enumerator from items

repeat

move enumerator to the next

deal with current value of enumerator

end

end

sentence print (message)

redirect to "Print"

end

phrase main

**create enumerable to** numbers

repeat with i from 1 to 10

print "Enumerating " & i

**yield return** i

end

end

**repeat with** number **in** numbers

if number >= 5

**break**

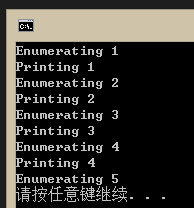
end

print "Printing " & number

end

end

什么叫模拟C#的yield return呢？就是连惰性计算也一起模拟！在main函数的第一部分，我创建了一个enumerable（iterator），包含1到10十个数字，而且每产生一个数字还会打印出一句话。但是接下来我在循环里面只取前5个，打印前4个，因此执行结果就是



当！

CPS风格的函数的威力在于，每一个函数都可以控制他如何执行函数结束之后写在后面的代码。也就是说，你可以根据你的需要，干脆选择保护现场，然后以后再回复。是不是听起来很像lua的coroutine呢？在Tinymoe，coroutine也可以自己做！

虽然函数最后被转换成了CPS风格的ast，而且测试用的生成C#代码的确也是原封不动的输出了出来，所以运行这个程序耗费了大量的函数调用。但这并不意味着Tinymoe的虚拟机也要这么做。大家要记住，一个语言也好，类库也好，给你的接口的概念，跟实现的概念，有可能完全不同。yield return写出来的确要花费点心思，所以《序言》我也不讲这么多了，后续的文章会详细介绍这方面的知识，当然了，还会告诉你怎么实现的。

## 尾声

这里我挑选了四个例子来展示Tinymoe最重要的一些概念。一门语言，要应用用起来简单，库写起来可以发挥想象力，才是有前途的。yield return例子里面的main函数一样，用的时候多清爽，清爽到让你完全忘记yield return实现的时候里面的各种麻烦的细节。

所以为什么我要挑选Tinymoe作为实例来科普编译原理呢？有两个原因。第一个原因是，想要实现Tinymoe，需要大量的知识。所以既然这个系列想让大家能够看完实现一个Tinymoe的低质量原型，当然会讲很多知识的。第二个原因是，我想通过这个例子向大家将一个道理，就是库和应用 、编译器和语法、实现和接口，完全可以做到隔离复杂，只留给最终用户简单的部分。**你看到的复杂的接口，并不意味着他的实现是臃肿的。你看到的简单的接口，也不意味着他的实现就很简洁**。

Tinymoe目前已经可以输出C#代码来执行了。后面我还会给Tinymoe加上静态分析和类型推导。对于这类语言做静态分析和类型推导又很多麻烦，我现在还没有完全搞明白。譬如说这种可以自己控制continuation的函数要怎么编译成状态机才能避免掉大量的函数调用，就不是一个容易的问题。所以在系列一边做的时候，我还会一边研究这个事情。如果到时候系列把编译部分写完的同时，这些问题我也搞明白的话，那我就会让这个系列扩展到包含静态分析和类型推导，继续往下讲。