陈卷毛 Lv1
2020年02月28日 阅读 821

关注

Wasm介绍之4：函数调用

上一篇文章介绍了WebAssembly（简称Wasm）内存和相关指令，这篇文章将介绍变量指令和函数调用指令。

全局变量

Wasm模块可以定义或者导入全局变量。导入时，可以限定全局变量的类型和可修改性（mutability）。定义时，除了限定类型和可修改性还可以给定初始值。下面是一个WAT例子，展示了全局变量的导入和定义：

复制代码

```
(module
  (import "env" "g1" (global $g1 i32))       ;; immutable
  (import "env" "g2" (global $g2 (mut f32))) ;; mutable
```

```
  (global $g5 f32 (f32.const 1.5))       ;; immutable
  (global $g6 f64 (f64.const 2.5))       ;; immutable

  (func $main
    ;; $g3 = $g1
    (global.get $g1)
    (global.set $g3)
  )
)
```

变量指令一共5条，其中2条用来读写全局变量，下面分别介绍。

global.get

`global.get` 指令（操作码 `0x23` ）把全局变量的值推入栈顶，全局变量的索引由指令的立即数参数（32位无符号整数）给定。下面是 `global.get` 指令的示意图：

复制代码

```
bytecode:
...][ global.get ][ global_idx ][...

stack:
|           |          |           |
|           |         ↘|globals[i] |
|     d     |          |     d     |
|     c     |          |     c     |
|     b     |          |     b     |
|     a     |          |     a     |
└───────────┘          └───────────┘
```

global.set

`global.set` 指令（操作码 `0x24` ）从栈顶弹出一个操作数，赋值给全局变量（弹出的操作数必须和全局变量类型相同）。和 `global.get` 指令一样，全局变量的索引也是由指令的立即数参数给定。下面是 `global.set` 指令的示意图：

复制代码

```
bytecode:
...][ global.set ][ global_idx ][...

stack:
|           |          |           |
```

```
|     b     |          |     b     |
|     a     |          |     a     |
└───────────┘          └───────────┘
```

局部变量

全局变量的作用域是整个Wasm模块，局部变量的作用域则是整个函数。下面是一个WAT例子，展示了局部变量的定义和使用：

复制代码

```
(module
  (func $main (param $a i32) (param $b f32)
    (local $c i32)
    (local $d i64)
    (local $e f32)
    (local $f f64)

    ;; $c = $a
    (local.get $a)
    (local.set $c)
  )
)
```

从上面的例子可以看到，函数的参数本质上其实就是局部变量。变量指令的其余3条用来读写局部变量，下面分别介绍。

local.get

`local.get` 指令（操作码 `0x20` ）和 `global.get` 指令类似，只不过读的是局部变量。下面是 `local.get` 指令的示意图：

复制代码

```
bytecode:
...][ local.get ][ local_idx ][...

stack:
|           |          |           |
|           |        ↘| locals[i] |
|     d     |          |     d     |
|     c     |          |     c     |
|     b     |          |     b     |
|     a     |          |     a     |
```

local.set

`local.set` 指令（操作码 `0x21` ）和 `global.set` 指令类似，只不过写的是局部变量。下面是 `local.set` 指令的示意图：

复制代码

```
bytecode:
...][ local.set ][ local_idx ][...

stack:
|           |          |           |
|           |          |           |
|     d     |↗         |           | # locals[local_idx] = d
|     c     |          |     c     |
|     b     |          |     b     |
|     a     |          |     a     |
└───────────┘          └───────────┘
```

local.tee

`local.tee` 指令（操作码 `0x22` ）指令和 `local.set` 指令类似，差别在于 `local.tee` 指令会把操作数留在栈顶。下面是 `local.tee` 指令的示意图：

复制代码

```
bytecode:
...][ local.tee ][ local_idx ][...

stack:
|           |          |           |
|           |          |           |
|     d     |↗        ↘|     d     | # locals[local_idx] = d
|     c     |          |     c     |
|     b     |          |     b     |
|     a     |          |     a     |
└───────────┘          └───────────┘
```

函数调用

函数调用指令属于控制指令，一共有两条： `call` 和 `call_indirect` 。本文只介绍 `call` 指令，`call_indirect` 和其余控制指令将在后续文章中介绍。

`call` 指令（操作码 `0x10` ）进行函数调用，函数索引由指令的立即数参数（32位无符号整数）指定。在执行该指令之前，需要把函数的参数准备好，按顺序放在栈顶（最左边的参数在最下面）。指令执行完毕后，参数已经从栈顶弹出，取而代之的是函数的返回值（如果有的话）。Wasm1.0规范规定函数的返回值最多不超过一个，后续版本可能会放开这个限制。下面是 `call` 指令的示意图（假设被调用函数接受两个 `i32` 类型的参数，返回一个 `i32` 类型的值）：

复制代码

```
bytecode:
...][ call ][ func_idx ][...

stack:
|           |          |           |
|           |          |           |
|     d     |↗         |           |
|     c     |↗        ↘|     r     | # funcs[func_idx](c,d)
|     b     |          |     b     |
|     a     |          |     a     |
└───────────┘          └───────────┘
```

下面的WAT例子展示了 `call` 、 `local.get` 、 `local.set` 等指令的用法：

复制代码

```
(module
  (func $main (export "main") (result i32)
    (call $max (i32.const 20) (i32.const 80))
  )
  (func $max (param $a i32) (param $b i32) (result i32)
    (local.get $a)
    (local.get $b)
    (i32.gt_s (local.get $a) (local.get $b))
    (select)
  )
)
```

*本文由CoinEx Chain开发团队成员Chase撰写。CoinEx Chain是全球首条基于Tendermint共识协议和Cosmos SDK开发的DEX专用公链，借助IBC来实现DEX公链、智能合约链、隐私链三条链合一的方式去解决可扩展性（Scalability）、去中心化（Decentralization）、安全性（security）区块链不可能三角的问题，能够高性能的支持数字资产的交易以及基于智能合约的Defi应用。

陈卷毛 Lv1
获得点赞 5 · 获得阅读 8,653

关注

输入评论（Enter换行，Ctrl + Enter发送）

发表评论

相关推荐

星星不懂前端啊o_o 24天前 WebAssembly 前端

初识 WASM

Wasm 是什么？ Wasm（WebAssembly） 是一种底层的汇编语言，能够在所有当代桌面浏览器及很多移动浏览器...

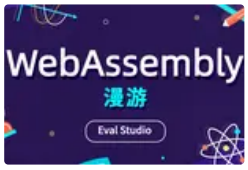
332 7 评论

EvalStudio 1月前 WebAssembly 前端

WebAssembly漫谈

WebAssembly是什么 —— https://webassembly.org/ 从官网释义来看，Wasm是基于栈...

643 31 评论



Sunbreak 1月前 Dart 前端

[Dart翻译]用Dart和Wasm做实验

将Dart编译为Wasm，并从Dart调用Wasm模块 WebAssembly（通常缩写为Wasm）是 "一种基于堆栈的虚拟机的...

173 5 评论

方石剑 1月前 Rust

WebAssembly会取代JavaScript吗？WASM 介绍和性能比较

WebAssembly是过去几十年来互联网最强大的创新之一，它是一个开放的标准，为可执行...

感谢 compose 函数，让我的代码屎山💩逐渐美丽了起来~

曾经有一段优美的代码放在我面前，我没有珍惜。直到现在每天面对代码屎山手足无措时，...

3.6w 781 152

sealyun 1月前 Go 前端 后端

rust+wasm写前端真香之路由

[sealer](https://github.com/alibaba/sealer)是阿里巴巴开源的基于kuberentes的集群镜像开源技术，可以把整个...

1674 7 评论

Michael_Yuan 1月前 Rust WebAssembly 前端

目前大火的 Jamstack 到底是什么?

这篇文章将带你了解 Jamstack 的概念以及开发范式。我们也将讨论 Rust 与 WebAssembl...

1136 11 评论

sealyun 1月前 Go 后端

rust+wasm写前端真香之嵌套与循环

[sealer](https://github.com/alibaba/sealer)是阿里巴巴开源的基于kuberentes的集群镜像开源技术，可以把整个...

111 2 评论

云的世界 1月前 前端 JavaScript

【干货】私藏的这些高级工具函数，你拥有几个?

用极其精简的代码，编写的高级工具函数，覆盖localStorage已使用空间，桌面通知，视频...

2.6w 592 76

飞书技术 1月前 SQLite Rust 前端

飞书WASM实践——SQLite篇

SQLite是一个跨平台的关系型数据库，广泛使用于客户端开发，飞书也使用SQLite作为数据持久化存储；同时为了方...

3841 30 4

Michael_Yuan 1月前 Go WebAssembly

用 WasmEdge 和 YoMo 对实时数据流进行 AI 推理

我们将向你展示如何为基于 Tensorflow 的图片识别创建 Rust 函数，将其编译为...

209 2 评论

Michael_Yuan 1月前 Go WebAssembly 后端

607 3 评论

Michael_Yuan 2月前 云原生

使用 Docker 工具管理WasmEdge 中的 WebAssembly 程序 |

开发者可以利用 DockerHub 和 CRI-O 等 Docker 工具在 WasmEdge 中部署、管理和运...

391 4 评论

天行无忌 2月前 JavaScript

Vue源码学习 | 4个实用的Javascript技巧

学习一门语言的一种非常有效的方法就是阅读该编程语言开发的优秀开源项目的源代码。...

12.9w 126 14

只要我E得够快 3月前 前端

前端展望-WebAssembly（wasm）技术入门

wasm技术帮助前端解决性能瓶颈，VR/图像视频编辑/3D游戏的希望。本文旨在了解其作用和使用方式。

496 4 评论

Rust_Magazine 4月前 Rust 后端

华为 | 基于 TVM Rust Runtime 和 WASM 沙箱运行 AI 模型 【Rust 中文精选 3 月刊】

基于TVM Rust Runtime和WASM沙箱运行AI模型 作者： 王辉 / 后期编辑： 张汉东 说明 本文介绍了一种WASM与...

956 3 评论

Patract 5月前 智能合约

访谈|探索 Wasm 合约的无限可能

**Wasm，即 WebAssembly，是一种用来补充 JS 在运行上不足的“低级”语言——基于二...

88 点赞 评论

设计稿智能生成代码 5月前 前端 WebAssembly

当 wasm 遇上数据处理

本文主要讲述了，如何快速的将一些已经成熟的算法迁移到 JavaScript 并在生产环境中部...

373 7 1

Try to do 5月前 Vue.js

vue slot 插槽里面传递事件 调用外层的函数

由于 有些页面比较类似 都是有相同的东西 我就想着用插槽的方法 去写个模板组件减少 代码重复量 也可以节省开发

sh22n 5月前 JavaScript

金三银四的前端社招面经

目前工作快四年，年后投了一波简历，这里整理了一下新鲜出炉的前端面经，需要的可以自取。 面试挺累人的，每...

6.4w 1196 215