---

link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=58 sentences=109, words=564

1.V8的内存垃圾回收

- 内存
- V8
- 垃圾回收

2.计算机模型

2.1 寄存器

- 是中央处理器的组成部分
- 寄存器是有限存储容量的高速存储部件
- 可以用来暂存指令、数据和地址
- 寄存器内的数据可用于执行算术和逻辑运算
- 寄存器内的地址可用于指向内存的某个位置

2.2 内存

- 随机存取存储器(Random Access Memory)也叫内存，英文缩写RAM
- RAM是与CPU直接交换数据的内部存储器
- RAM工作时可以从任何一个指定地址写入或读出信息
- RAM用来在计算机中用来暂时存储程序、数据和中间结果

```
console.log(0xFFFFFFFF);
console.log(0xFFFFFFFF / 1024 / 1024 / 1024);
```

2.2.1 内存空间分类

- 内存空间 (https://www.processon.com/diagraming/61b2e22863768956b5f64f68)会分为二类
    - 数据空间
    - 指令空间

2.3 指令

- 可以通过指令指挥命令计算机进行工作

2.3.1 机器语言指令

- 计算机只认识0和1，所以我们可以通过二进制指令和计算机进行沟通
- 这些指令被称为指令集，也就是机器语言
- MIPS是一种采取精简指令集(RISC)的处理器架构
- 最常见的MIPS-32位指令集每个指令是一个32位的二进制数

2.3.2 汇编指令

- 二进制指令难以编写和阅读，所以出现了汇编指令集
-
    - MIPS Assembler and Runtime Simulator (http://courses.missouristate.edu/KenVollmar/MARS/)是一个轻量级的交互式开发环境IDE,用于使用MIPS(Microprocessor without interlocked piped stages architecture)汇编语言进行编程

部分 值 含义 opcode操作码 0 add做加法运算 rs $s1 第一个来源寄存器 rt $s2 第二个来源寄存器 rd $s3 目标寄存器 shamt 0 位移量 funct 32 函数,这个字段选择Opcode操作某个特定变体 助记符 opcode rs rt rd shamt funct 示例 示例含义 操作 add 000000 rs rt rd 00000 100000 add $1,$2,$3 $1=$2+$3 带符号数相加 addu 000000 rs rt rd 00000 100001 add $1,$2,$3 $1=$2+$3 无符号数相加 sll 000000 00000 rt rd sa 000000 sll rd,rt,sa rd = rt << sa 逻辑左移,寄存器rt的值向左移sa位，结果保存到rd寄存器中

```
add $s3,$s1,$s2
```

```
addi $s1,$zero,1
sll s2,s1,1
```

```
console.log((0x02329820).toString('2'));
```

2.4 程序指针

- PC寄存器存储着下一条要执行的指令的内存地址
- 机器循环=PC位置获取指令->分析指令->执行指令->PC指针移动
- 执行过程 (https://www.processon.com/diagraming/61b2e2f8e401fd0ae56242ab)

汇编指令 含义 addi $s1,$zero,1 把$zero中的值加上数字1保存到寄存器$1中 addi $s2,$zero,2 把$zero中的值加上数字2保存到寄存器$1中 add $s3,$s1,$s2 把寄存器$s1和寄存器$2的和存储到$s3中

3.V8

- V8引擎 (https://gitee.com/zhufengpeixun/v8/)是一个JavaScript引擎实现

3.1 语言的分类

3.1.1 解释执行

- 先将源代码通过解析器转成中间代码，再用解释器执行中间代码，输出结果
- 启动快，执行慢

3.1.2 编译执行

- 先将源代码通过解析器转成中间代码，再用编译器把中间代码转成机器码，最后执行机器码，输出结果
- 启动慢，执行快

3.2 V8执行过程

- V8采用的是解释和编译两种方式，这种混合使用的方式称为JIT技术
- 第一步先由解析器生成抽象语法树和相关的作用域
- 第二步根据AST和作用域生成字节码，字节码是介于AST和机器码的中间代码
- 然后由解释器直接执行字节码，也可以让编译器把字节码编译成机器码后再执行
- jsvu (https://github.com/GoogleChromeLabs/jsvu)可以快速安装V8引擎
- V8源码编译出来的可执行程序名为d8 d8 (https://v8.dev/docs/d8),d8是V8自己的开发工具shell

3.2.1 抽象语法树

- astexplorer (https://astexplorer.net/)可以查看抽象语法树

```
var a = 1;
var b = 2;
var c = a + b;
```

```
d8 --print-ast 4.js
[generating bytecode for function: ]
--- AST ---
FUNC at 0
. KIND 0
. SUSPEND COUNT 0
. NAME ""
. INFERRED NAME ""
. DECLS
. . VARIABLE (00000278B965EB88) (mode = VAR) "a"
. . VARIABLE (00000278B965EC78) (mode = VAR) "b"
. . VARIABLE (00000278B965EDB0) (mode = VAR) "c"
. BLOCK NOCOMPLETIONS at -1
. . EXPRESSION STATEMENT at 8
. . . INIT at 8
. . . . VAR PROXY unallocated (00000278B965EB88) (mode = VAR) "a"
. . . . LITERAL 1
. BLOCK NOCOMPLETIONS at -1
. . EXPRESSION STATEMENT at 20
. . . INIT at 20
. . . . VAR PROXY unallocated (00000278B965EC78) (mode = VAR) "b"
. . . . LITERAL 2
. BLOCK NOCOMPLETIONS at -1
. . EXPRESSION STATEMENT at 32
. . . INIT at 32
. . . . VAR PROXY unallocated (00000278B965EDB0) (mode = VAR) "c"
. . . . ADD at 34
. . . . . VAR PROXY unallocated (00000278B965EB88) (mode = VAR) "a"
. . . . . VAR PROXY unallocated (00000278B965EC78) (mode = VAR) "b"
```

3.2.2 作用域

- 作用域是一个抽象的概念，它描述了一个变量的生命周期，比如哪些变量是在哪里声明的，哪些变量是在哪里使用的

```
d8 --print-scopes 1.js
Global scope:
global {

  TEMPORARY .result;

  VAR c;
  VAR b;
  VAR a;
}
```

3.2.3 字节码

- 字节码是机器码的抽象表示
- 源代码直接编译成机器码编译时间太长，体积太大，不适合移动端
- 编译成字节码编译时间短，体积小
- bytecodes.h (https://gitee.com/zhufengpeixun/v8/blob/master/src/interpreter/bytecodes.h)
- FeedBack Vector slot(反馈向量槽)是一个数组，是用来给优化编译器提供信息的
- 字节码执行过程 (https://www.processon.com/diagraming/61b37c957d9c086a6766b72f)

```
var a = 10;
var b = 20;
var c = a + b;
```

```
d8 --print-bytecode  4.js
LdaConstant [0] 从常量池中加载索引0的常量到累加寄存器中
Star r1 把累加器的值保存到目标寄存器中
LdaZero 把0保存到累加寄存器中
Star r2 把累加器的值0保存到目标寄存器中
Mov , r3 保存r3寄存器的值
CallRuntime [DeclareGlobals], r1-r3

StackCheck    检查栈是否溢出

LdaSmi [10]    加载10到累加寄存器中
StaGlobal [1] 把累加寄存器的值保存到常量池索引1处

LdaSmi [20]    加载20到累加寄存器中
StaGlobal [2] 把累加寄存器的值保存到常量池索引2处

LdaGlobal [1] 从常量池加载索引1到累加寄存器
Star r1        把累加器的值10保存到目标计数器中
LdaGlobal [2] 从常量池加载索引2的值20到累加寄存器
Add r1  把r1寄存器的值加到累加寄存器中，累加寄存器值为30
StaGlobal [3] 把累加寄存器的值保存到常量池索引3处
LdaUndefined  把Undefined保存到累加寄存器中
Return         返回累加寄存器中的值
```

3.2.4 编译器优化

```
function sum() {
    let a = 1;
    let b = 2;
    return a + b;
}
for (let i = 0; i < 10000; i++) {
    sum();
}
```

```
d8 --trace-opt sum.js
[marking 0x02ccc2ba2279  for optimized recompilation, reason: small function, ICs with typeinfo: 4/4 (100%), generic ICs: 0/4 (0%)]
[marking 0x02ccc2ba2339  for optimized recompilation, reason: small function, ICs with typeinfo: 1/1 (100%), generic ICs: 0/1 (0%)]
[compiling method 0x02ccc2ba2339  using TurboFan]
```