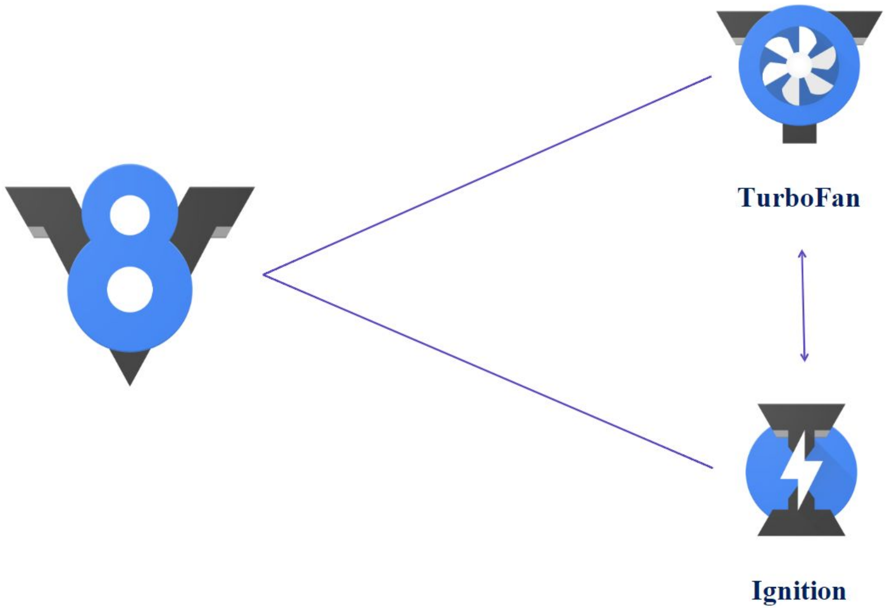
V8中的指针压缩及其源码分析

陆亚涵
PLCT Lab 工程师
yahan@iscas.ac.cn
2020.12

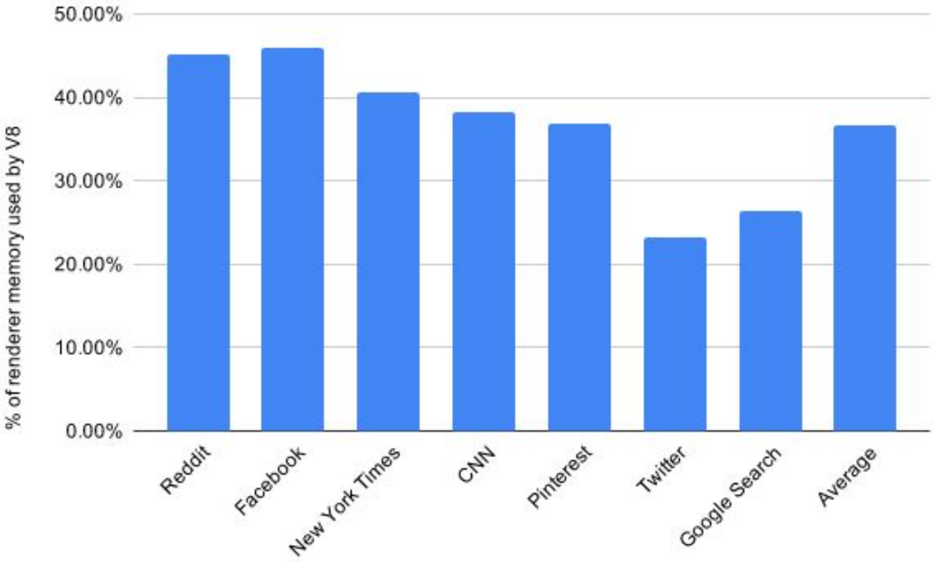
目录

关于V8



V8 引擎是用 C ++编写的开源高性能 JavaScript 和 WebAssembly 引擎，它已被用于 Chrome 和 Node.js 等.

Pointer-compressed 背景



Chrome浏览不同页面V8消耗的内存占比
https://v8.dev/blog/pointer-compression

2014年,Chrome浏览器由32位切换为64位，这给浏览器带来更好的安全性、稳定性和性能，但与此同时也带来了内存消耗的增加：指针占用的内存由原来的4字节变为8字节。

而V8中所有的Object都是分配在Heap中，故每个Object都有一个指针指向该Object.

Value tagging

V8 中任何变量(objects, arrays, numbers or strings),都是存储在Heap中的,因此即使是int型变量,也需要一个指针.为了降低指针带来的内存消耗,V8采用了Tagged pointer技术.

Tagged pointer利用指针无论是32位架构还是64位架构,低2bit的值总是0的特点,将一些tag存储在低2bit中.V8利用tag来区别是否是Pointer还是Smi,Pointer是Weak还是Strong.

```
                  |----- 32 bits -----|
Pointer:          |_____address_____w1|
Smi:              |___int31_value____0|
```

```
            |----- 32 bits -----|----- 32 bits -----|
Pointer:    |________________address______________w1|
Smi:        |____int32_value____|0000000000000000000|
```

上图: 32位处理器架构的Value tagging
下图: 64位处理器架构的Value tagging
来源:https://v8.dev/blog/pointer-compression

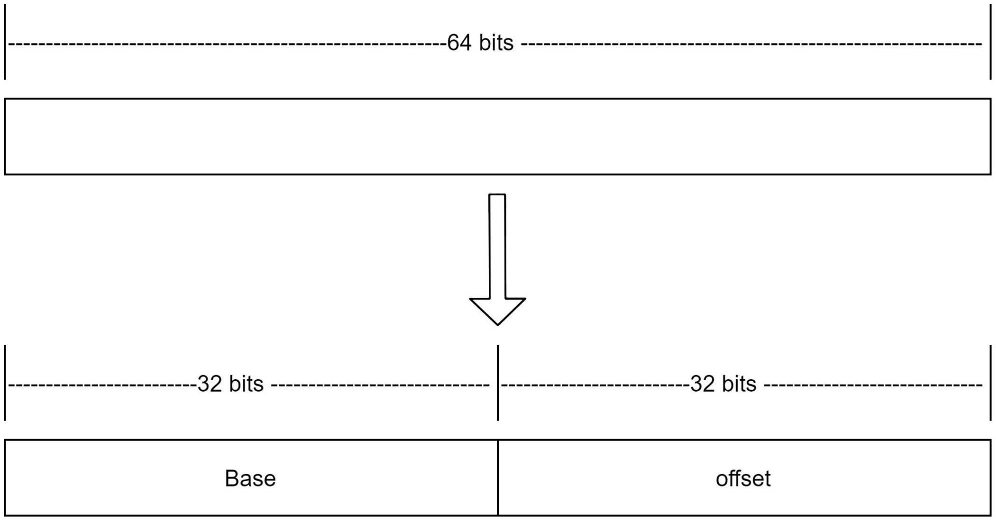
Pointer-compressed 基本原理

Pointer-Compressed 是谷歌用来减少内存占用的方法之一，原理：

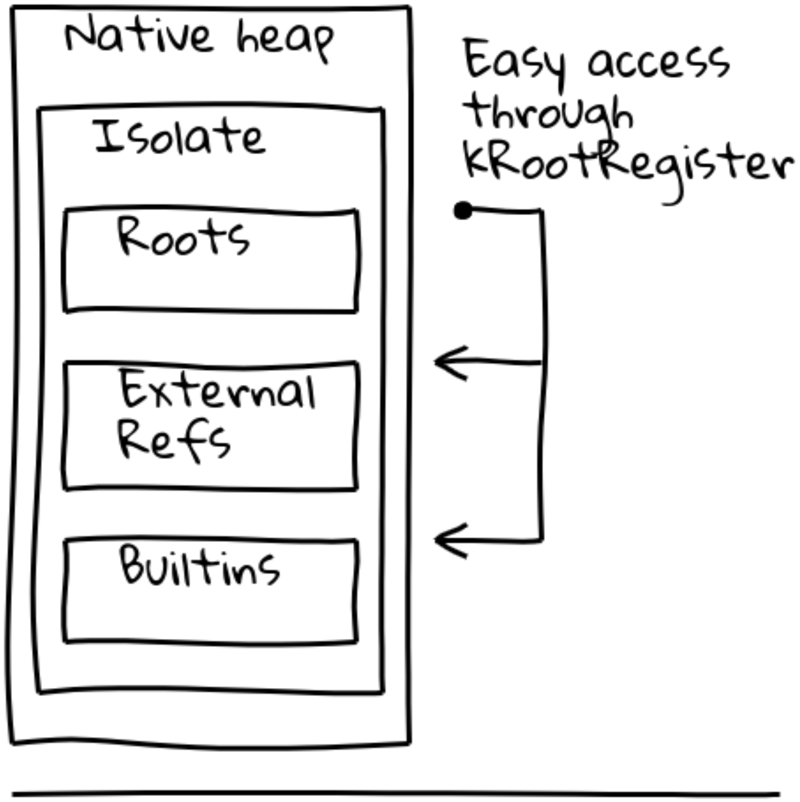
- 采用32位的offset和32bit base来代替64位的指针
- base由一个全局变量持有，指针只需存储offset到内存中

为了适配指针压缩,需要满足以下两个条件：

- 所有 V8 objects 都要分配在4GB范围中
- 将指针用offset重新解释,需要一个全局变量保存base



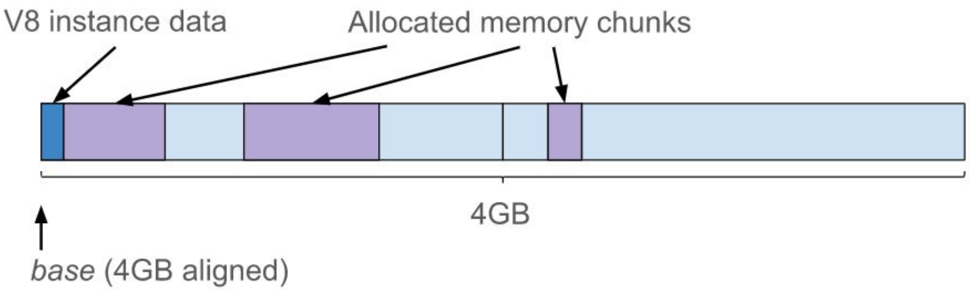
V8的先天优势



V8的isloate布局

来源:https://v8.dev/blog/pointer-compression

Pointer-compress下内存布局



Base对齐到4GB内存的起始位置
https://v8.dev/blog/pointer-compression

```
uint64_t uncompressed_tagged;
uint32_t compressed_tagged = uint32_t(uncompressed_tagged);
```

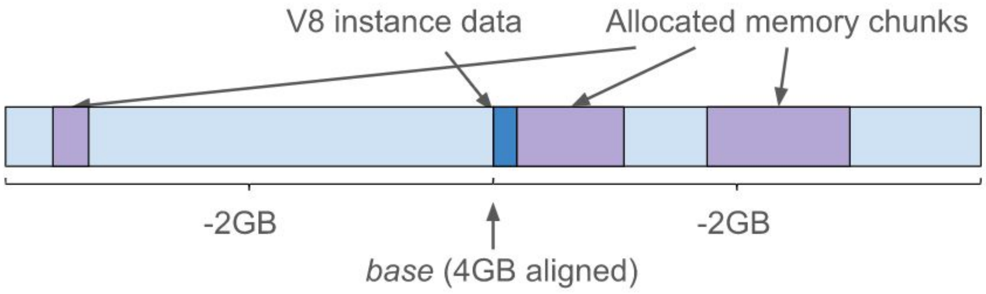
compression

```
uint32_t compressed_tagged;

uint64_t uncompressed_tagged;
if (compressed_tagged & 1) {
  // pointer case
  uncompressed_tagged = base + uint64_t(compressed_tagged);
} else {
  // Smi case
  uncompressed_tagged = int64_t(compressed_tagged);
}
```

decompression

Pointer-compress下几种内存布局



Base对齐到4GB内存的中间位置
https://v8.dev/blog/pointer-compression

```
int32_t compressed_tagged;

// Common code for both pointer and Smi cases
int64_t uncompressed_tagged = int64_t(compressed_tagged);
if (uncompressed_tagged & 1) {
  // pointer case
  uncompressed_tagged += base;
}
```

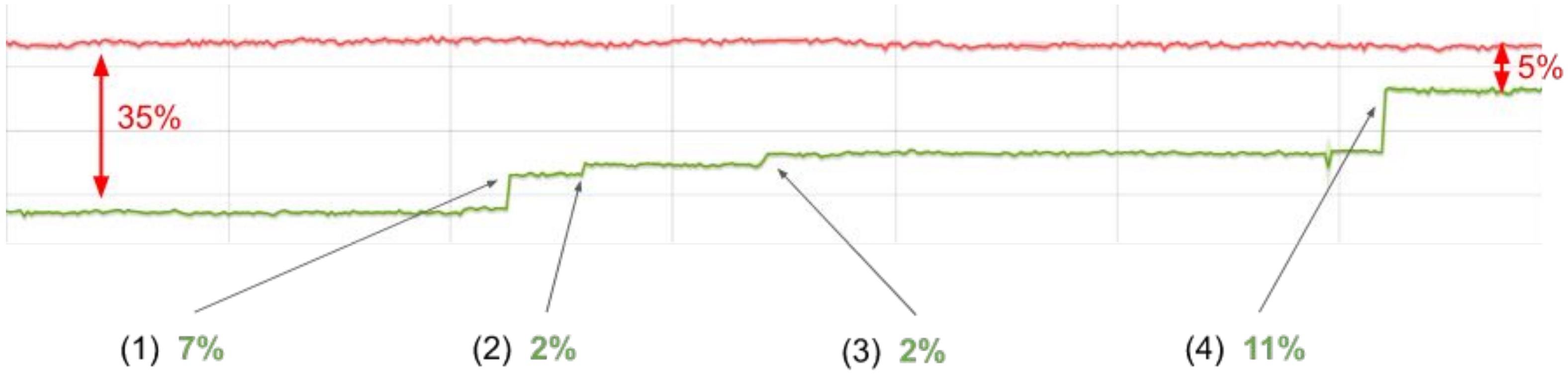
decompression

```
int32_t compressed_tagged;

// Same code for both pointer and Smi cases
int64_t sign_extended_tagged = int64_t(compressed_tagged);
int64_t selector_mask = -(sign_extended_tagged & 1);
// Mask is 0 in case of Smi or all 1s in case of pointer
int64_t uncompressed_tagged =
    sign_extended_tagged + (base & selector_mask);
```

无分支的decompression

Pointer-compress带来的性能损失及其优化



Octane’s score on x64 architecture
https://v8.dev/blog/pointer-compression

优化一　Branchful version was 7% faster on x64

Decompression	Branchless	Branchful

Code	`movsxlq r11,[…]`
	`movl r10,r11`
	`andl r10,0x1`
	`negq r10`
	`andq r10,r13`
	`addq r11,r10`
		`movsxlq r11,[…]`
		`testb r11,0x1`
		`jz done`
		`addq r11,r13`
		`done:`

Summary	20 bytes	13 bytes
	6 instructions executed	3 or 4 instructions executed
	no branches	1 branch
	1 additional register	

X64下decompression汇编代码对比
https://v8.dev/blog/pointer-compression

Decompression	Branchless	Branchful

Code	`ldur w6, […]`
	`sbfx x16, x6, #0, #1`
	`and x16, x16, x26`
	`add x6, x16, w6, sxtw`
		`ldur w6, […]`
		`sxtw x6, w6`
		`tbz w6, #0, #done`
		`add x6, x26, x6`
		`done:`

Summary	16 bytes	16 bytes
	4 instructions executed	3 or 4 instructions executed
	no branches	1 branch
	1 additional register	

ARM64下decompression汇编代码对比
https://v8.dev/blog/pointer-compression

同样Arm64下有分支的代码同样比无分支代码快

现代CPU分支预测技术已经十分强大，影响类似这样的代码执行的效率主要取决于代码的执行指令数量或code size。

优化二: Eliminate decompressions directly followed by compressions 2%

```
Reduction DecompressionElimination::ReduceCompress(Node* node) {
  DCHECK(IrOpcode::IsCompressOpcode(node->opcode()));

  DCHECK_EQ(node->InputCount(), 1);
  Node* input_node = node->InputAt(0);
  IrOpcode::Value input_opcode = input_node->opcode();
  if (IrOpcode::IsDecompressOpcode(input_opcode)) {
    DCHECK(IsValidDecompress(node->opcode(), input_opcode));
    DCHECK_EQ(input_node->InputCount(), 1);
    return Replace(input_node->InputAt(0));
  } else if (IsReducibleConstantOpcode(input_opcode)) {
    return Replace(GetCompressedConstant(input_node));
  } else {
    return NoChange();
  }
}
```

优化三: 去除多余的指令 2%

```
movl rax, <mem>   // load
movlsxlq rax, rax // sign extend
```

→

```
movlsxlq rax, <mem>
```

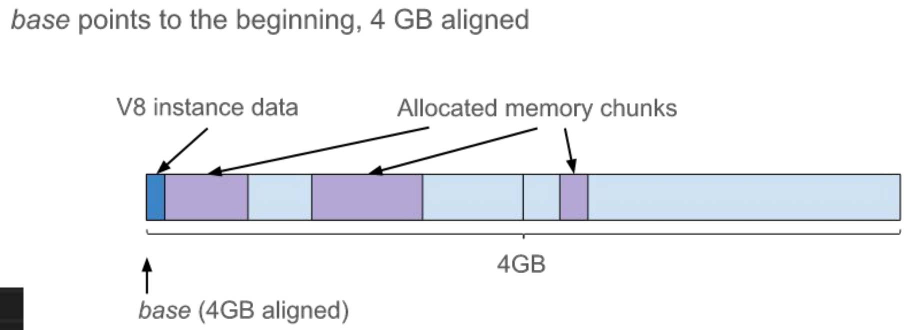
优化四: Updated the pattern matching 与 Decompressed Optimize 11%

优化五: Smi-corrupting 2.5%

```
int64_t uncompressed_tagged = int64_t(compressed_tagged);
if (uncompressed_tagged & 1) {
  // pointer case
  uncompressed_tagged += base;
}
```

⇩

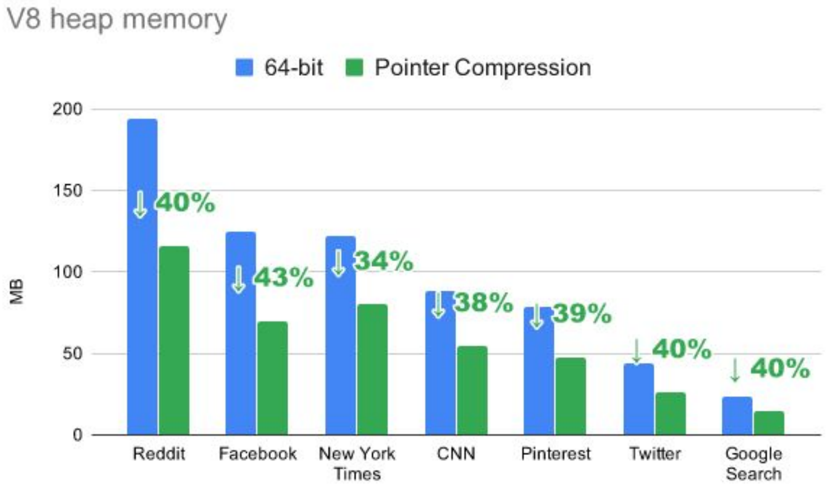
```
int64_t uncompressed_tagged = base + int64_t(compressed_tagged);
```



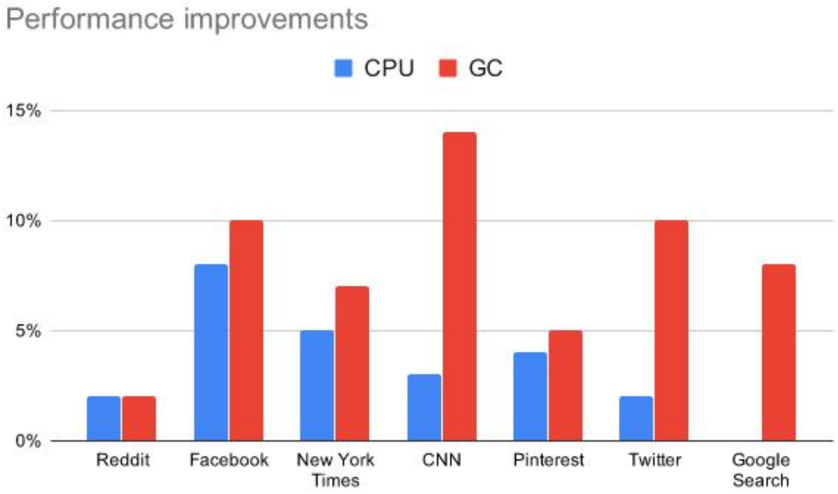
Base对齐到4GB内存的起始位置
https://v8.dev/blog/pointer-compression

若采用的内存布局是 base对齐到4GB内存空间的起始位置时,int64_t可变为uint64_t,将符号扩展变为零扩展,进一步提高性能

结果



采用指针压缩前后内存对比
https://v8.dev/blog/pointer-compression



采用指针压缩后CPU和GC的性能提升
https://v8.dev/blog/pointer-compression