Node Profiler

一款JavaScript的性能调优工具

powered by

自我介绍

- 朴灵@阿里云/alinode团队
- JacksonTian@GitHub
- 《深入浅出Node.js》作者
- 目前从事Node/V8开发及相关技术产品开发

Agenda

- 什么是Node Profiler
- Node Profiler的工作原理
- 怎么使用Node Profiler

Node Profiler

- 来自阿里云的alinode团队的性能调优工具
- 基于Node进行开发，hack了部分V8代码
- 完全兼容Node，集成改进的inspector工具

常用的性能调优工具

- benchmark.js
- node-webkit-agent/chrome dev tools
- ab/wrk
- --prof & mac-tick-processor

mac-tick-processor

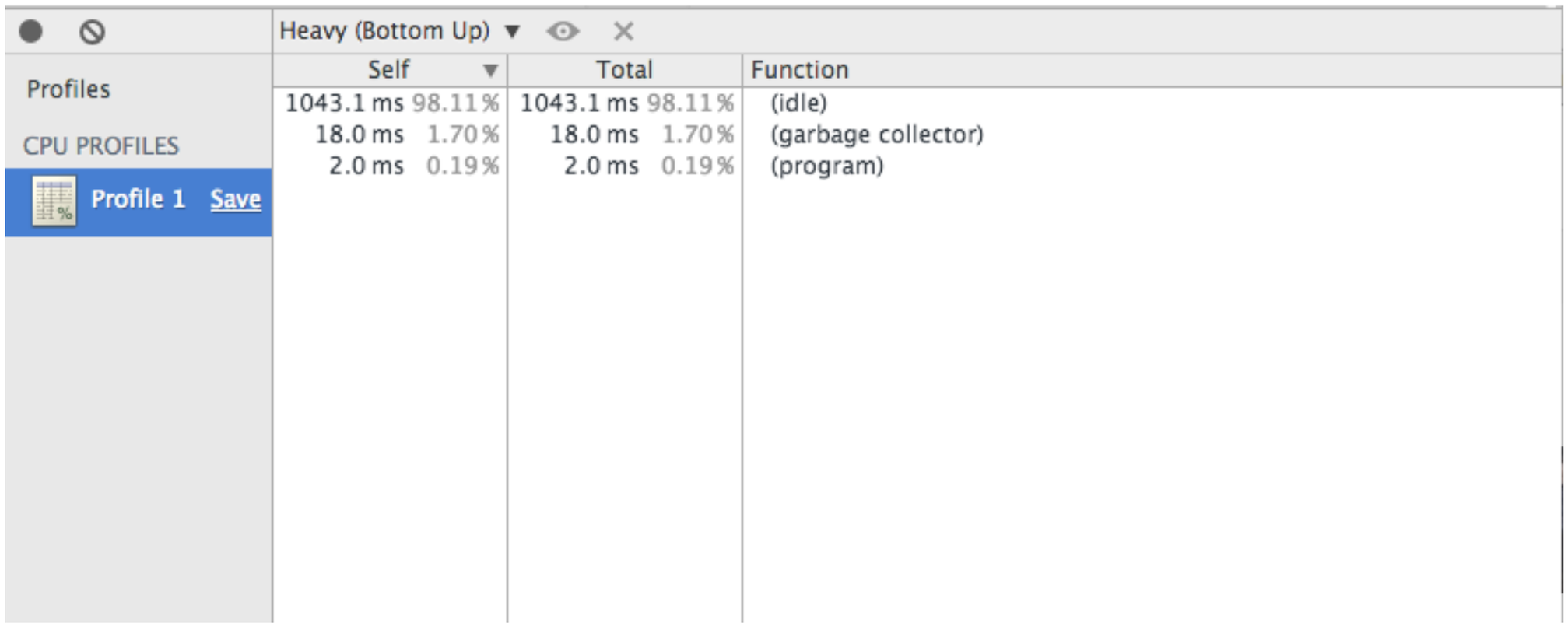
```
$ ~/git/v8/tools/mac-tick-processor isolate-0x102006200-v8.log
Statistical profiling result from isolate-0x102006200-v8.log, (51 ticks, 2
unaccounted, 0 excluded).

 [Shared libraries]:
   ticks  total  nonlib   name
      1    2.0%          /usr/lib/system/libsystem_platform.dylib

 [JavaScript]:
   ticks  total  nonlib   name
      1    2.0%    2.0%  LazyCompile: ~NativeModule.compile node.js:836:44
      1    2.0%    2.0%  LazyCompile: ~EventEmitter events.js:25:22

 [C++]:
   ticks  total  nonlib   name
     15   29.4%   30.0%
node::ContextifyScript::New(v8::FunctionCallbackInfo<v8::Value> const&)
      3    5.9%    6.0%  node::SetupProcessObject(node::Environment*, int, char
const* const*, int, char const* const*)
      2    3.9%    4.0%  _malloc_zone_from_ptr
      2    3.9%    4.0%  __simple_dprintf
      2    3.9%    4.0%  ___mac_set_link
      1    2.0%    2.0%  v8::internal::Zone::NewExpand(int)
      1    2.0%    2.0%  v8::internal::VariableProxy*
v8::internal::Scope::NewUnresolved<v8::internal::AstConstructionVisitor>(v8::int
ernal::AstNodeFactory<v8::internal::AstConstructionVisitor>*,
v8::internal::AstRawString const*, v8::internal::Interface*, int)
      1    2.0%    2.0%
v8::internal::TemplateHashMapImpl<v8::internal::ZoneAllocationPolicy>::Lookup(vo
id*, unsigned int, bool, v8::internal::ZoneAllocationPolicy)
```

chrome dev tools



benchmark.js

```
$ node apply_call.js
call_method() x 32,516,593 ops/sec ±2.11% (81 runs sampled)
apply_method() x 23,153,561 ops/sec ±1.66% (84 runs sampled)
strict_call_method() x 33,664,441 ops/sec ±1.96% (90 runs sampled)
strict_apply_method() x 40,285,700 ops/sec ±2.56% (86 runs sampled)
Fastest is strict_apply_method()
```

常见调优工具的问题

- 很容易知道哪些代码慢，但较少知道原因。
- Node Profiler的目标是不仅要知道哪些代码，还要知道为什么，以及更多。

V8知多少

- V8是一个JavaScript语言的执行引擎
- V8是JIT的方式执行JavaScript代码，即：将JavaScript直接编译为机器码，然后执行
- V8对JavaScript的处理是以函数为单位进行的

编译结果

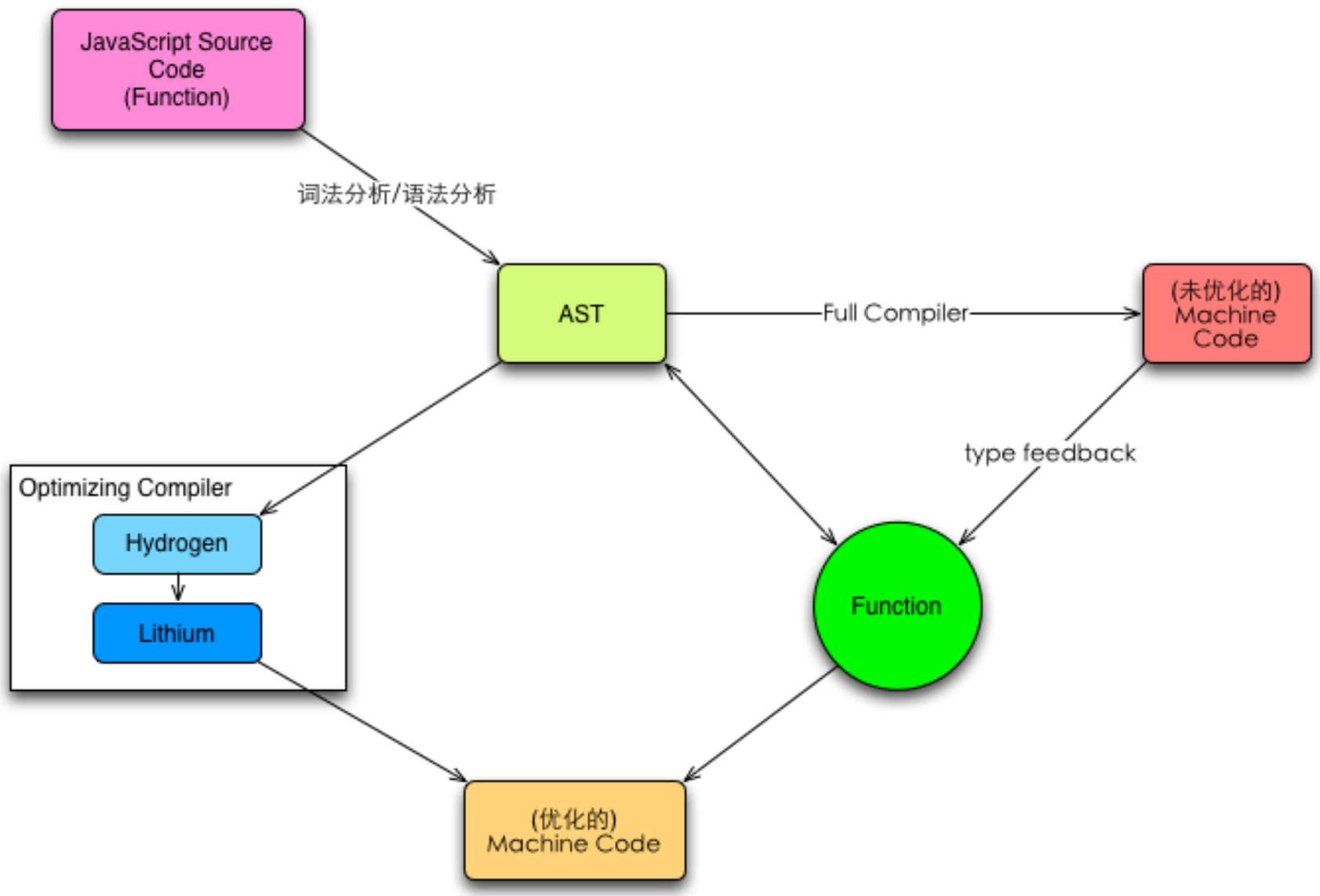
```
(a, b) {
  return a + b;
}
```

```
0x39803a80d160     0  488b4c2418     REX.W movq rcx,[rsp+0x18]
0x39803a80d165     5  493b4da8       REX.W cmpq rcx,[r13-0x58]
0x39803a80d169     9  750d           jnz 24  (0x39803a80d178)
0x39803a80d16b    11  488b4e27       REX.W movq rcx,[rsi+0x27]
0x39803a80d16f    15  488b492f       REX.W movq rcx,[rcx+0x2f]
0x39803a80d173    19  48894c2418     REX.W movq [rsp+0x18],rcx
0x39803a80d178    24  55             push rbp
0x39803a80d179    25  4889e5         REX.W movq rbp,rsp
0x39803a80d17c    28  56             push rsi
0x39803a80d17d    29  57             push rdi
0x39803a80d17e    30  493ba598070000 REX.W cmpq rsp,[r13+0x798]
0x39803a80d185    37  7305           jnc 44  (0x39803a80d18c)
0x39803a80d187    39  e8945bf2ff     call StackCheck  (0x39803a732d20)    ;; debug: statement 167
                                     ;; code: BUILTIN
0x39803a80d18c    44  ff7518         push [rbp+0x18]
0x39803a80d18f    47  488b4510       REX.W movq rax,[rbp+0x10]
0x39803a80d193    51  5a             pop rdx
0x39803a80d194    52  e8c7c9f0ff     call 0x39803a719b60     ;; debug: statement 178
                                     ;; debug: position 187
                                     ;; code: BINARY_OP_IC, UNINITIALIZED (id = 8)
0x39803a80d199    57  90             nop
0x39803a80d19a    58  48bba160b0df3c020000 REX.W movq rbx,0x23cdfb060a1    ;; object: 0x23cdfb060a1 Cell fo
0x39803a80d1a4    68  83430bd1       addl [rbx+0xb],0xd1
0x39803a80d1a8    72  791f           jns 105  (0x39803a80d1c9)
0x39803a80d1aa    74  50             push rax
0x39803a80d1ab    75  e8f05bf2ff     call InterruptCheck  (0x39803a732da0)    ;; code: BUILTIN
0x39803a80d1b0    80  58             pop rax
0x39803a80d1b1    81  48bba160b0df3c020000 REX.W movq rbx,0x23cdfb060a1    ;; object: 0x23cdfb060a1 Cell fo
0x39803a80d1bb    91  49ba0000000000180000 REX.W movq r10,0x180000000000
0x39803a80d1c5   101  4c895307       REX.W movq [rbx+0x7],r10
0x39803a80d1c9   105  488be5         REX.W movq rsp,rbp      ;; debug: statement 192
                                     ;; js return
0x39803a80d1cc   108  5d             pop rbp
0x39803a80d1cd   109  c21800         ret 0x18
```

Crankshaft

- 一个普通编译器：FullCompiler
- 一个优化编译器：Optimizing Compiler
- 运行时优化

V8优化过程


JavaScript Source Code (Function)
词法分析/语法分析
AST
Full Compiler
(未优化的) Machine Code
type feedback
Optimizing Compiler
Hydrogen
Lithium
Function
(优化的) Machine Code

优化的成果

```
var add = function (a, b) {
  return a + b;
}
```

node --print_unopt_code --print_opt_code --always_opt example.js

	指令数
优化前	132
优化后	98

注：衡量CPU性能的指标之一：MIPS——每秒百万条定点指令

事与愿违

- DONT_OPTIMIZE_NODE
- Bailout
- deoptimization(逆优化)

Node Profiler的改进

- 了解更多函数状态
- 给出更多优化建议

How to use it

- install node-profiler from http://alinode.aliyun.com/
- node-profiler example.js # 运行起来
- wrk http://localhost:1334/ # 让代码燃
- start profiling/stop profiling # 采样
- analyse profiling result # 分析结果

live demo

相关术语

- self：函数自身执行时间
- total：函数自身及所调用函数的执行时间
- # of hidden classes created：函数执行过程中创建的隐藏类数量
- Reason for deoptimization: 未优化的原因
- Function：函数
- opt: 优化次数
- deopt：逆优化
- monomorphic：单态
- polymorphic: 多态

优化指南

- 只优化瓶颈代码，不要优化无关的
- Node Profiler仅对Node(V8)有效，不保证其他环境的有效性
- Node Profiler仅对CPU层面有优化，系统性能跟很多（其他）因素相关
- 不要使用在生产环境中！！！

TODO

- release node-profiler for iojs.
- add more bailout cases.
- open source it.

“process.exit(0);”

–Jackson Tian