Inside sablejs

打造更快更安全的 JavaScript 实现

赵洋

sablejs 作者

赵洋

前百度、腾讯、全民直播前端工程师

前 GMTC、GIAC、FDCon 等多个会议讲师

专栏作者、开源爱好者、Web 技术拥护者

https://github.com/ErosZy

大纲

- 起因
- 目标
- 优化思路及方法
  - 函数调用开销及消除
  - 编译期优化
  - 对象属性访问优化
- 总结与展望
  - sablejs 2.0

低危风险

通过滑动验证识别用户为低危风险后，仅需一步即可完成所有验证流程，整体验证时间低于1秒。

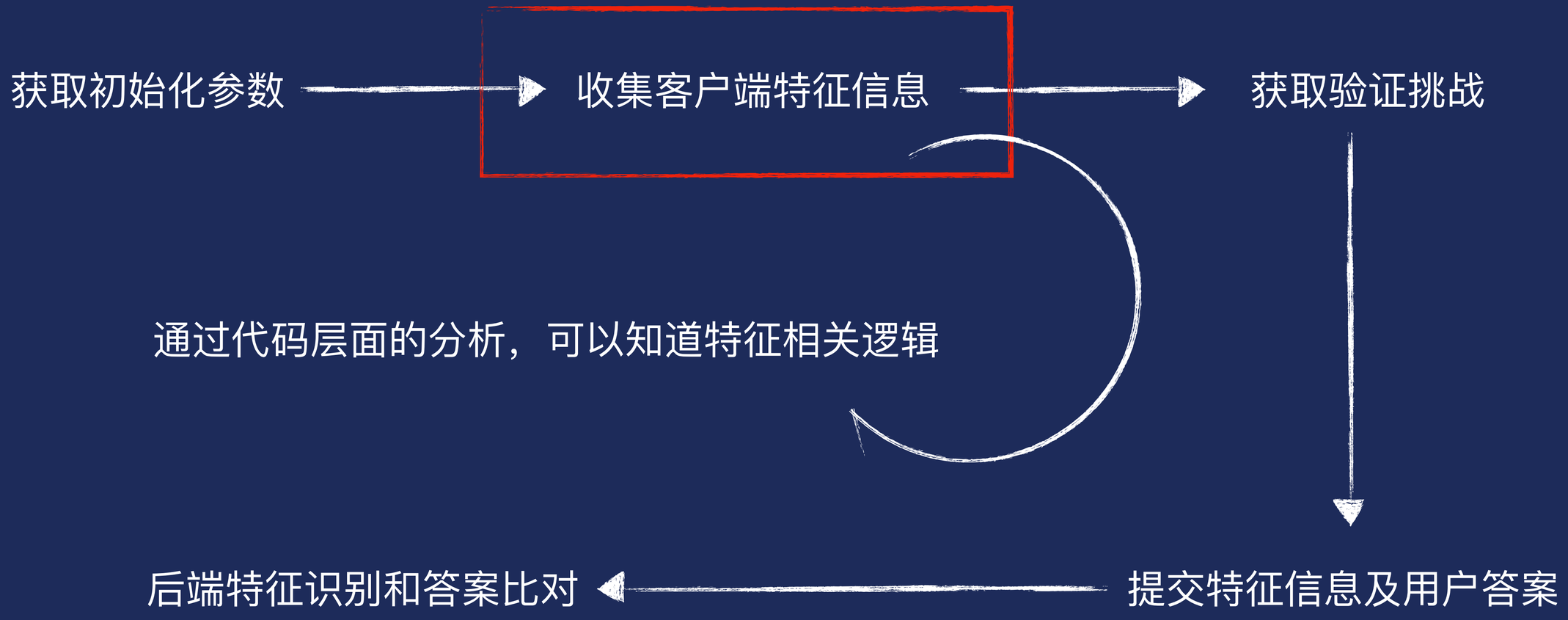
中危风险

高危风险

智能判断

友验 基于虚拟机保护、设备特征识别、AI 行为判断等多项功能打造的全面可靠的验证体系

友验 基于虚拟机保护、设备特征识别、AI行为判断等多项功能打造的全面可靠的验证体系


获取初始化参数
收集客户端特征信息
获取验证挑战
通过代码层面的分析，可以知道特征相关逻辑
后端特征识别和答案比对
提交特征信息及用户答案

如何实现虚拟机保护?

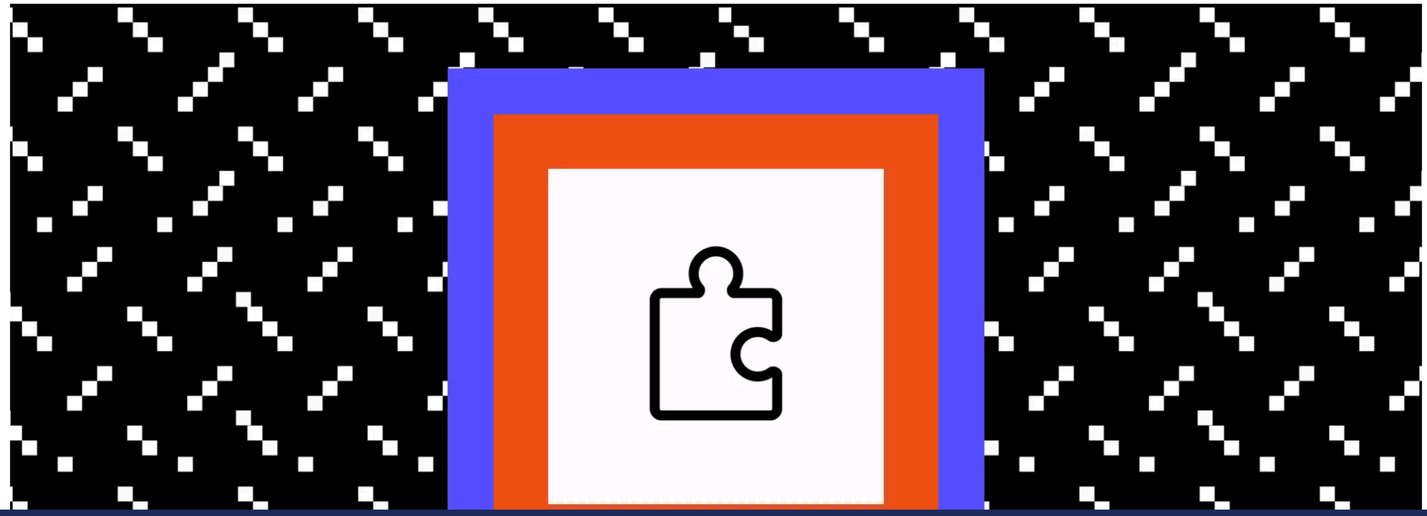
WebAssembly

WebAssembly + QuickJS

How to build a plugin system on the web and also sleep well at night

Rudi Chen
Software Engineer, Figma

August 22, 2019



JavaScript ← eval — WebAssembly

互相调用过程中并不安全 🤦

JavaScript In JavaScript

eval5、sval、sandboxjs....

性能太差、无test262单测覆盖率、玩具实现

_TENCENT_VM_（腾讯）、_$jsvmprt（字节）

性能太差、VM 初始化时间过长

性能太差、VM 初始化时间过长

加密逻辑无法在 VM 内部执行，外部执行逻辑直接暴露

性能太差、VM 初始化时间过长

影响页面相关性能


sablejs

The safer and faster ECMA5.1 interpreter written by JavaScript

1. Sandbox(easy and better than Figma)
2. Mini Program/Game dynamic execution
3. Browser JavaScript source code projection

不服跑个分?

```
function fib(n) {
    if(n <= 1) return n;
    else return fib(n-1) + fib(n-2);
}

fib(35)

// sable.js: 5872ms
// jerryscript: 4933ms
// mujs: 5749ms
```

```
def fib(n)
    if n <= 1 then
        return n;
    else
        return fib(n-1) + fib(n-2)
    end
end

fib(35)

// Ruby 1.8.7: 7892ms
// Ruby 2.7.2: 1939ms
// Ruby 3.0.0: 1430ms
```

	sablejs	sval	eval5	quickjs-wasm	goja
Language	JavaScript	JavaScript	JavaScript	C + WebAssembly	Golang
Richards	110	24.9	24.7	376	208
Crypto	114	24.6	20.2	400	104
RayTrace	258	92.2	98.5	471	294
NavierStokes	183	35.9	49.8	665	191
DeltaBlue	120	35.3	29.5	402	276
Total score	148	37.3	37.3	452	202
Baseline	1	▼ 2.96	▼ 2.96	▲ 2.05	▲ 0.36
File Size(KB)	216	152	134	434	-
Gzip Size(KB)	29	40	34	245	-

型号	类型	VM初始化(ms)	DOM节点attach(ms)	执行总耗时(ms)
PC(AMD Ryzen 5 3600)	中高端	53	3	56
iPhone XS	高端	40	5	45
iPhone 6	中低端	165	11	176
OPPO Reno	中高端	237	8	245
三星S6edge	中低端	489	22	511
小米4A	低端	506	21	527
小米2S	低端	896	43	939

sablejs 在同类实现中性能优异

如何做到的?

1. 函数调用开销及消除

2. 一些编译期优化（计算操作与作用域）

3. 对象属性访问的优化

4. and more...

以上优化思路均来自各语言 VM 实现

函数调用开销及消除

```
exit: for (;;) {
  try {
    switch (opcode_1881[J_1872.pc++]) {
      case 0:…
        }
      case 1:…
        }
      case 2:…
        }
      case 3:…
        }
      case 4:…
        }
      case 5:…
        }
      case 6:…
        }
      case 7:…
        }
      case 8:…
        }
      case 9:…
        }
      case 10:…
        }
      case 11:…
        }
      case 12:…
        }
      case 13:…
        }
```

```
__case(function E_pop(J, func, opcode) { // 0      ErosZy, 3 months ago • [UPD] add Date/Error functions
  __pop(J, 1);
  break;
});

__case(function E_dup(J, func, opcode) { // 1
  __dup(J);
  break;
});

__case(function E_dup2(J, func, opcode) { // 2
  __dup2(J);
  break;
});

__case(function E_rot2(J, func, opcode) { // 3
  __rot2(J);
  break;
});

__case(function E_rot3(J, func, opcode) { // 4
  __rot3(J);
  break;
});

__case(function E_rot4(J, func, opcode) { // 5
  __rot4(J);
  break;
});

__case(function E_int(J, func, opcode) { // 6
  __pushNumber(J, numtab[opcode[J.pc++]]);
  break;
});

__case(function E_num(J, func, opcode) { // 7
  __pushNumber(J, numtab[opcode[J.pc++]]);
  break;
});

__case(function E_literal(J, func, opcode) { // 8
  __pushString(J, strtab[opcode[J.pc++]]);
  break;
});
```

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

```
function add(a, b, c) {
    Code:
        0: load 0
        1: load 1
        2: add
        3: load 2
        4: add
        5: return
}
```

分支预测大失败！

V8 JIT 生成成本 > 执行收益

大量指令函数没有被优化

函数级别 JIT -> 语句级别 JIT

“避免函数调用” - Inline Function

macro

```
function R_strictEq(J) {
  var lValue = __stackidx(J, -2);
  var rValue = __stackidx(J, -1);
  if (__isString(lValue) && __isString(rValue)) {
    return V_toString(J, lValue) === V_toString(J, rValue);
  } else if (__isUndefined(lValue) && __isUndefined(rValue)) {
    return true;
  }

  if (lValue.type !== rValue.type) {
    return false;
  } else if (__isUndefined(lValue)) {
    return true;
  } else if (__isNull(lValue)) {
    return true;
  } else if (__isNumber(lValue)) {
    return lValue.value === rValue.value;
  } else if (__isBoolean(lValue)) {
    return lValue.value === rValue.value;
  } else if (__isObject(lValue)) {
    return lValue.value === rValue.value;
  }
}
```

```
function R_strictEq(J_808) {
  var lValue_809 = J_808.stack[J_808.top + -2];
  var rValue_810 = J_808.stack[J_808.top + -1];
  if (lValue_809.type === 5 && rValue_810.type === 5) {
    return V_toString(J_808, lValue_809) === V_toString(J_808, rValue_810);
  } else if ((lValue_809.type === 1 || lValue_809.type === 0) && (rValue_810.type === 1 ||
rValue_810.type === 0)) {
    return true;
  }
  if (lValue_809.type !== rValue_810.type) {
    return false;
  } else if (lValue_809.type === 1 || lValue_809.type === 0) {
    return true;
  } else if (lValue_809.type === 2) {
    return true;
  } else if (lValue_809.type === 4) {
    return lValue_809.value === rValue_810.value;
  } else if (lValue_809.type === 3) {
    return lValue_809.value === rValue_810.value;
  } else if (lValue_809.type === 6) {
    return lValue_809.value === rValue_810.value;
  }
}$
```

Build your dream language

```
syntax __isArray = ctx => {
  const values = unwrap(ctx.next().value).value;
  const object = values.get(1);
  return #`(${object}.type === 6 && ${object}.value.type === 8)`;
};
```

Faster JavaScript calls

Published 15 February 2021 · Tagged with internals

JavaScript allows calling a function with a different number of arguments than the expected number of parameters, i.e., one can pass fewer or more arguments than the declared formal parameters. The former case is called under-application and the latter is called over-application.

In the under-application case, the remaining parameters get assigned the undefined value. In the over-application case, the remaining arguments can be accessed by using the rest parameter and the `arguments` property, or they are simply superfluous and they can be ignored. Many Web/Node.js frameworks nowadays use this JS feature to accept optional parameters and create a more flexible API.

Until recently, V8 had a special machinery to deal with arguments size mismatch: the arguments adaptor frame. Unfortunately, argument adaption comes at a performance cost, but is commonly needed in modern front-end and middleware frameworks. It turns out that, with a clever trick, we can remove this extra frame, simplify the V8 codebase and get rid of almost the entire overhead.

We can calculate the performance impact of removing the arguments adaptor frame through a micro-benchmark.

```
console.time();
function f(x, y, z) {}
for (let i = 0; i <  N; i++) {
  f(1, 2, 3, 4, 5);
}
console.timeEnd();
```

```
Overhead  Symbol
  17.02%  *doQuickSort ../dist/source-map.js:2752
  11.20%  Builtin:ArgumentsAdaptorTrampoline
   7.17%  *compareByOriginalPositions ../dist/source-map.js:1024
   4.49%  Builtin:CallFunction_ReceiverIsNullOrUndefined
   3.58%  *compareByGeneratedPositionsDeflated ../dist/source-map.js:1063
   2.73%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   2.11%  Builtin:StringEqual
   1.93%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.66%  *doQuickSort ../dist/source-map.js:2752
   1.25%  v8::internal::StringTable::LookupStringIfExists_NoAllocate
   1.22%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   1.21%  Builtin:StringCharAt
   1.16%  Builtin:Call_ReceiverIsNullOrUndefined
   1.14%  v8::internal::(anonymous namespace)::StringTableNoAllocateKey::IsMatch
   0.90%  Builtin:StringPrototypeSlice
   0.86%  Builtin:KeyedLoadIC_Megamorphic
   0.82%  v8::internal::(anonymous namespace)::MakeStringThin
   0.80%  v8::internal::(anonymous namespace)::CopyObjectToObjectElements
   0.76%  v8::internal::Scavenger::ScavengeObject
   0.72%  v8::internal::String::VisitFlat<v8::internal::IteratingStringHasher>
   0.68%  *SourceMapConsumer_parseMappings ../dist/source-map.js:1894
   0.64%  *doQuickSort ../dist/source-map.js:2752
   0.56%  v8::internal::IncrementalMarking::RecordWriteSlow
```

Thanks, QuickJS !

编译期优化

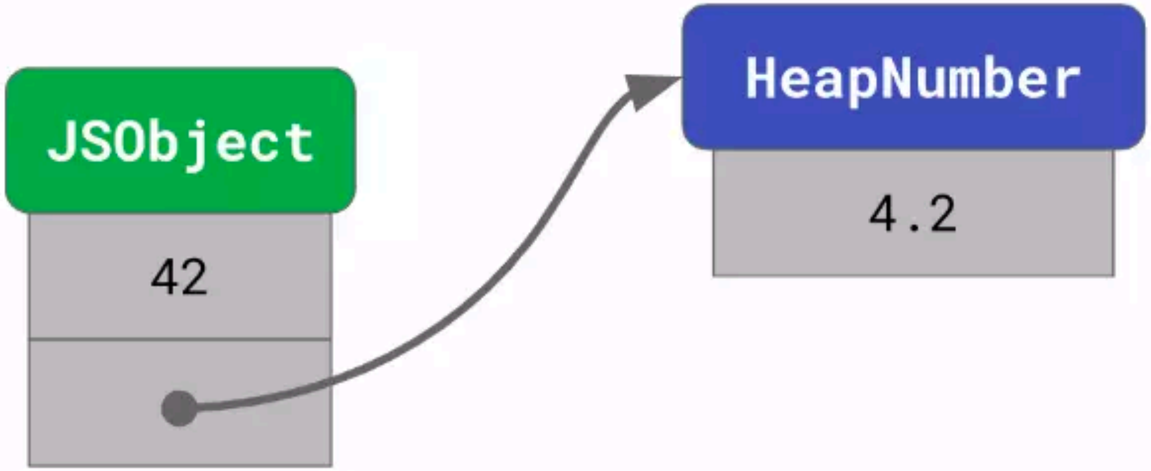
计算开销

Types are always ‘Boxed’

SMI and HeapNumber

```
 -Infinity // HeapNumber
-(2**30)-1 // HeapNumber
  -(2**30) // Smi
        -42 // Smi
         -0 // HeapNumber
          0 // Smi
        4.2 // HeapNumber
         42 // Smi
    2**30-1 // Smi
      2**30 // HeapNumber
   Infinity // HeapNumber
        NaN // HeapNumber
```

```
o = {
  x: 42,
  y: 4.2,
};
```



快速整数操作

1 + 1

“1” + a

```
{} + []
```

11.6.1 The Addition operator (+)

The addition operator either performs string concatenation or numeric addition.

The production *AdditiveExpression* : *AdditiveExpression* + *MultiplicativeExpression* is evaluated as follows:

1. Let *lref* be the result of evaluating AdditiveExpression.
2. Let *lval* be GetValue(*lref*).
3. Let *rref* be the result of evaluating MultiplicativeExpression.
4. Let *rval* be GetValue(*rref*).
5. Let *lprim* be ToPrimitive(*lval*).
6. Let *rprim* be ToPrimitive(*rval*).
7. If Type(*lprim*) is String or Type(*rprim*) is String, then
    a. Return the String that is the result of concatenating ToString(*lprim*) followed by ToString(*rprim*)
8. Return the result of applying the addition operation to ToNumber(*lprim*) and ToNumber(*rprim*). See the Note below 11.6.3.

NOTE 1 No hint is provided in the calls to ToPrimitive in steps 5 and 6. All native ECMAScript objects except Date objects handle the absence of a hint as if the hint Number were given; Date objects handle the absence of a hint as if the hint String were given. Host objects may handle the absence of a hint in some other manner.

NOTE 2 Step 7 differs from step 3 of the comparison algorithm for the relational operators (11.8.5), by using the logical-or operation instead of the logical-and operation.

```
function R_concat(J) {
  var lValue = V_toPrimitive(J, __stackidx(J, -2), HINT.None);
  var rValue = V_toPrimitive(J, __stackidx(J, -1), HINT.None);
  if (__isString(lValue) || __isString(rValue)) {
    __pop(J, 2);
    __pushString(J, V_toString(J, lValue) + V_toString(J, rValue));
  } else {
    __pop(J, 2);
    __pushNumber(J, V_toNumber(J, lValue) + V_toNumber(J, rValue));
  }
}
```

$$1 + 1 = 2$$

```
function R_concat(J) {
  var lValue = V_toPrimitive(J, __stackidx(J, -2), HINT.None);
  var rValue = V_toPrimitive(J, __stackidx(J, -1), HINT.None);
  if (__isString(lValue) || __isString(rValue)) {
    __pop(J, 2);
    __pushString(J, V_toString(J, lValue) + V_toString(J, rValue));
  } else {
    __pop(J, 2);
    __pushNumber(J, V_toNumber(J, lValue) + V_toNumber(J, rValue));
  }
}
```

“1” + a

```
function R_concat(J) {
  var lValue = V_toPrimitive(J, __stackidx(J, -2), HINT.None);
  var rValue = V_toPrimitive(J, __stackidx(J, -1), HINT.None);
  if (__isString(lValue) || __isString(rValue)) {
    __pop(J, 2);
    __pushString(J, V_toString(J, lValue) + V_toString(J, rValue));
  } else {
    __pop(J, 2);
    __pushNumber(J, V_toNumber(J, lValue) + V_toNumber(J, rValue));
  }
}
```

{} + []

```
function R_concat(J) {
  var lValue = V_toPrimitive(J, __stackidx(J, -2), HINT.None);
  var rValue = V_toPrimitive(J, __stackidx(J, -1), HINT.None);
  if (__isString(lValue) || __isString(rValue)) {
    __pop(J, 2);
    __pushString(J, V_toString(J, lValue) + V_toString(J, rValue));
  } else {
    __pop(J, 2);
    __pushNumber(J, V_toNumber(J, lValue) + V_toNumber(J, rValue));
  }
}
```

```
__case(OP_MUL | VAR | CONST_N, function(J, func, opcode) {…
});

__case(OP_MUL | CONST_N | VAR, function(J, func, opcode) {…
});

__case(OP_DIV | VAR | CONST_N, function(J, func, opcode) {…
});

__case(OP_DIV | CONST_N | VAR, function(J, func, opcode) {…
});

__case(OP_MOD | VAR | CONST_N, function(J, func, opcode) {…
});

__case(OP_MOD | CONST_N | VAR, function(J, func, opcode) {…
});

__case(OP_ADD | VAR | CONST_N, function(J, func, opcode) {…
});
```

Thanks, PHP !

作用域优化

```
// 直接运行
var a = 0;
for(var i = 0; i < 10000000; i++){
    a += 1;
}

// QuickJS: 540ms

// 闭包运行
(function(){
    var a = 0;
    for(var i = 0; i < 10000000; i++){
        a += 1;
    }
})()

// QuickJS: 90ms

// 注意，较新的JS VM已经针对这些case做了独立优化，均不再存在此类问题
```

```
// 直接运行
var a = 0;
for(var i = 0; i < 10000000; i++){
    a += 1;
}

// QuickJS: 540ms

// 闭包运行
(function(){
    var a = 0;
    for(var i = 0; i < 10000000; i++){
        a += 1;
    }
})()

// QuickJS: 90ms

// 注意，较新的JS VM已经针对这些case做了独立优化，均不再存在此类问题
```

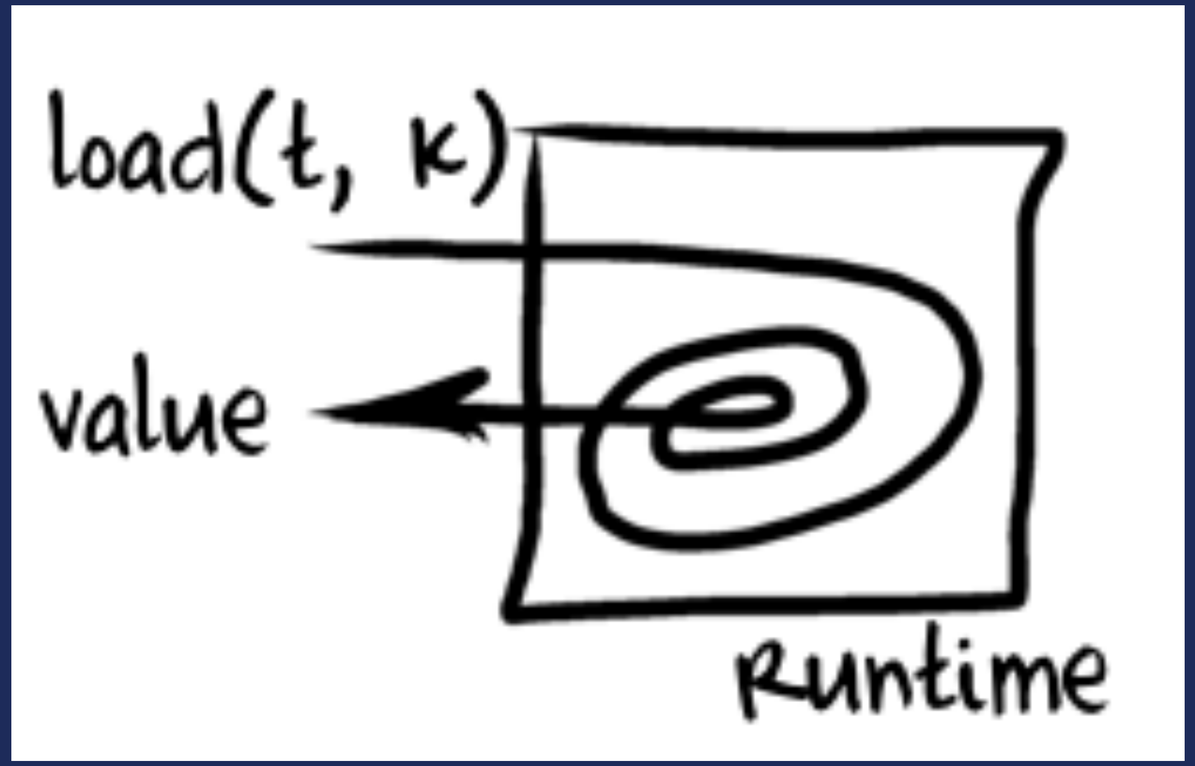
```
// 直接运行
var a = 0;
for(var i = 0; i < 10000000; i++){
    a += 1;
}

// QuickJS: 540ms

// 闭包运行
(function(){
    var a = 0;
    for(var i = 0; i < 10000000; i++){
        a += 1;
    }
})()

// QuickJS: 90ms

// 注意，较新的JS VM已经针对这些case做了独立优化，均不再存在此类问题
```

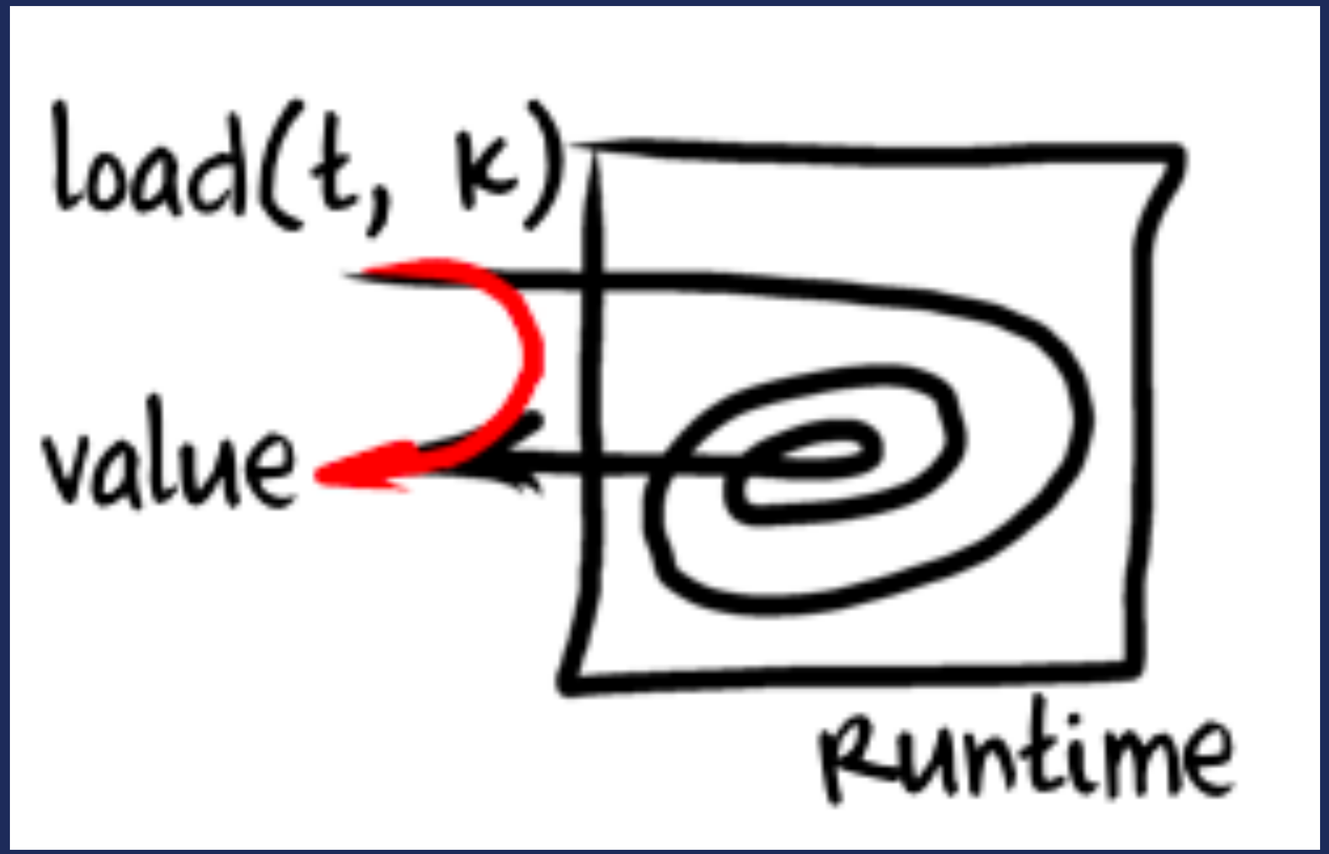
编译阶段尽可能找出可以分配在栈上的变量

Thanks, Lua and Ruby !

对象属性访问优化

Inline Cache


load(t, k)
value
Runtime


load(t, k)
value
Runtime

737ms  117ms

Thanks, V8 !

Future Plan

sablejs 2.0

```
(function(){
  for(var i = 0; i < 10000000; i++);
}());

// sablejs 2.0: 276.279ms --- baseline
// sablejs 1.0.6 878ms --- slower: 218.11%
// quickjs-wasm: 228ms --- faster: 17.39%
```

```
for (;;) {
  switch (opcode) {
    case 0: {
      // ...
    }
    case 1: {
      // ...
    }
    case 2: {
      // ...
    }
  }
}
```

```
function add() {
  return 1 + 2 + 3;
}
```

```
function __C_add(J) {
  __pushNumber(J, 1);
  __pushNumber(J, 2);
  __R_add(J);
  __pushNumber(J, 3);
  __R_add(J);
}
```

Thanks, DoppioJVM !

QCon+ 案例研习社

扫码学习大厂案例

学习前沿案例，向行业领先迈进

40 个
热门专题
—
行业专家把关内容筹备，
助你快速掌握最新技术发展趋势

200 个
实战案例
—
了解大厂前沿实战案例，
为 200 个真问题找到最优解

40 场
直播答疑
—
40 位技术大咖，每周分享最新
技术认知，互动答疑

365 天
持续学习
—
视频结合配套 PPT
畅学 365 天

GMTC
全球大前端技术大会

THANKS

InfoQ