《Chrome V8 源码》34. 终级优化技术 Turbofan

1 摘要

Turbofan 是基于 Sea of Nodes 理论的优化编译，它是 V8 Compiler Pipeline 三个节点中的最后一个节点，此外还有 Ingintion 和 Sparkplug。 Turbofan 使 JavaScript 执行的更快，但也需要更多的编译时间，所以 V8 只对热点函数使用 Turbofan。本文将介绍 Turbofan 的开启方式、工作流程以及重要的数据结构。

2 Turbofan

下面给出测试样例代码：

```
function addstring(a,b) {
    return a+b;
}
addstring("hello ","Turbofan!");
//分隔线.........................
Bytecode Age: 0
         00000043637A1DDE @    0 : 13 00             LdaConstant [0]
         00000043637A1DE0 @    2 : c2                Star1
         00000043637A1DE1 @    3 : 19 fe f8          Mov <closure>, r2
         00000043637A1DE4 @    6 : 64 51 01 f9 02    CallRuntime [DeclareGlobals],
r1-r2
         00000043637A1DE9 @   11 : 21 01 00          LdaGlobal [1], [0]
         00000043637A1DEC @   14 : c2                Star1
         00000043637A1DED @   15 : 13 02             LdaConstant [2]
```

```
          00000043637A1DEF @   17 : c1                Star2
          00000043637A1DF0 @   18 : 13 03             LdaConstant [3]
          00000043637A1DF2 @   20 : c0                Star3
          00000043637A1DF3 @   21 : 62 f9 f8 f7 02    CallUndefinedReceiver2 r1,
r2, r3, [2]
          00000043637A1DF8 @   26 : c3                Star0
          00000043637A1DF9 @   27 : a8                Return
Constant pool (size = 4)
00000043637A1D79: [FixedArray] in OldSpace
 - map: 0x01d0222812c1 <Map>
 - length: 4
           0: 0x0043637a1d01 <FixedArray[2]>
           1: 0x0043637a1c09 <String[9]: #addstring>
           2: 0x0043637a1c29 <String[6]: #hello >
           3: 0x0043637a1c41 <String[9]: #Turbofan!>
```

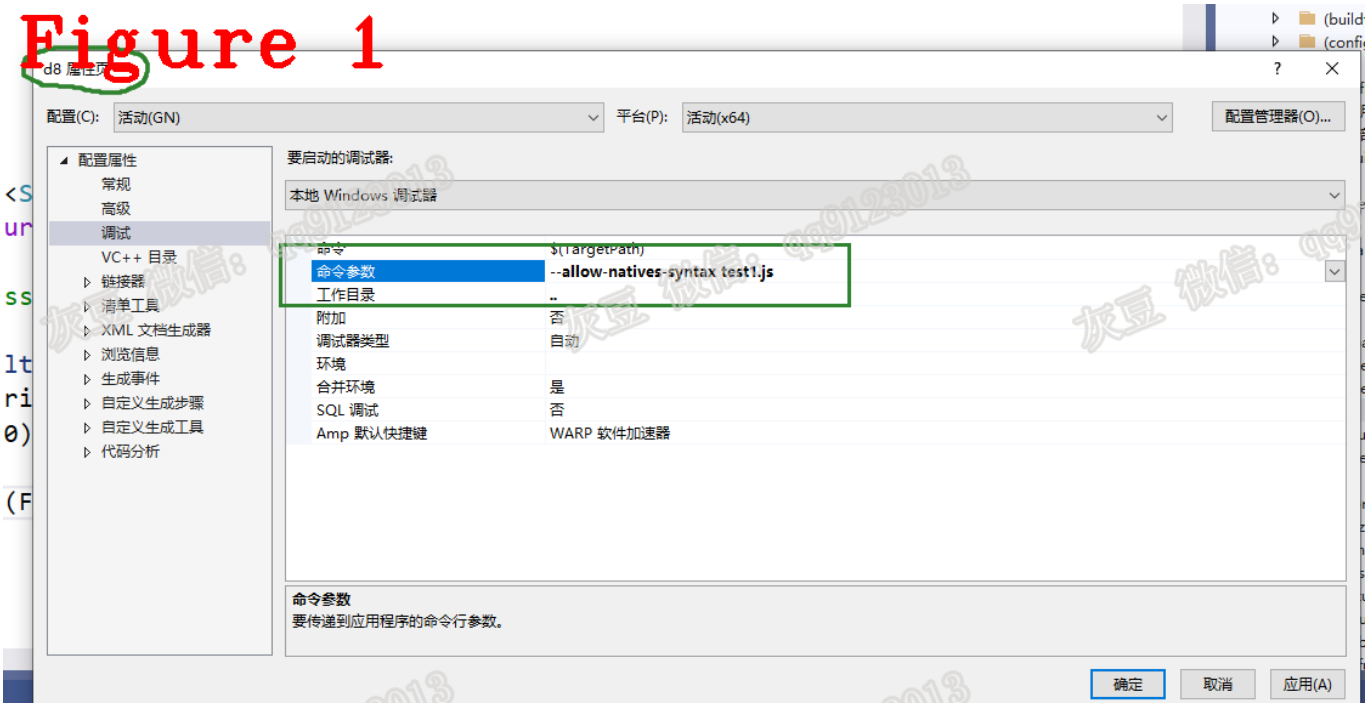
上述代码中 LdaGlobal [1], [0]、LdaConstant [2] 和 LdaConstant [3] 三条指令分别获取 addstring() 函数和两个参数；CallUndefinedReceiver2 调用 addstring() 函数完成字符串相加。 只有热点函数才会启动 Turbofan，为了方便学习，需要使用 %OptimizeFunctionOnNextCall() 和 --allow-natives-syntax 指令主动触发 Turbofan。

(1) %OptimizeFunctionOnNextCall() 用于在下次调用时启动优化编译；

(2) --allow-natives-syntax 允许 d8 接受 natives 指令。图 1 给出了使用方法。

(3) %DebugPrint 用于打印调试信息。

Figure 1



对测试代码进行一些改动，源码如下：

```
1.  function addstring(a,b) {
2.      return a+b;
3.  }
4.  console.log(addstring("hello ","Turbofan!"));
5.  %DebugPrint(addstring);
6.  %OptimizeFunctionOnNextCall(addstring);
7.  addstring("Speculative ", "Optimization");
8.  %DebugPrint(addstring);
```

上述第 4 行代码执行后会产生 Feedback；第 5 行代码使用 Turbofan 优化 addstring；Turbofan 优化机制的核心思想是投机，即会省去很多不必要的操作来提升性能，简而言之 Turbofan 认为 addstring 只用于字符串的加法操作，期望下次调用 addstring 时传入的两个参数仍然是字符串，这样就可以省去很多不必要的操作。执行第 5 行代码后打印的信息如下：

```
hello Turbofan!
DebugPrint: 000000F3196E1FA1: [Function] in OldSpace
 - map: 0x00ae91d413a1 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
 - elements: 0x03e492c81309 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype:
 - initial_map:
 - shared_info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
 - name: 0x00f3196e1c09 <String[9]: #addstring>
 - builtin: InterpreterEntryTrampoline
 - formal_parameter_count: 2
 - kind: NormalFunction
 - context: 0x00f3196c34a9 <NativeContext[262]>
 - code: 0x021668546441 <Code BUILTIN InterpreterEntryTrampoline>
 - interpreted
 - bytecode: 0x00f3196e20c1 <BytecodeArray[6]>
 - source code: (a,b) {
     return a+b;
}
//.........省略.....................
 - feedback vector: No feedback vector, but we have a closure feedback cell
array
000003E492C82FB9: [ClosureFeedbackCellArray] in ReadOnlySpace
 - map: 0x03e492c81f19 <Map>
 - length: 0
000000AE91D413A1: [Map]
 - type: JS_FUNCTION_TYPE
 - instance size: 64
 - inobject properties: 0
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - stable_map
 - callable
 - constructor
 - has_prototype_slot
 - back pointer: 0x03e492c81599 <undefined>
 - prototype_validity cell: 0x01cfdce84a09 <Cell value= 1>
 - instance descriptors (own) #5: 0x00f3196c4041 <DescriptorArray[5]>
 - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
 - constructor: 0x00f3196c3fd9 <JSFunction Function (sfi = 000001CFDCE91409)>
 - dependent code: 0x03e492c81239 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 - construction counter: 0
```

上述信息是执行 Turbofan 之前的 debug 输出，第 1 行是运行结果；第 8 行是 SharedFunctionInfo；第 10 行说明启动方式为 InterpreterEntryTrampoline，也就是解释执行；第 14 行是 InterpreterEntryTrampoline 的入口地址；第 23-42 行给出了 addstring 的 map 信息。
下面给出执行 Turbofan 之后的 debug 信息：

```
1.  Speculative Optimization
2.  DebugPrint: 000000F3196E1FA1: [Function] in OldSpace
3.   - map: 0x00ae91d413a1 <Map(HOLEY_ELEMENTS)> [FastProperties]
4.   - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
5.   - elements: 0x03e492c81309 <FixedArray[0]> [HOLEY_ELEMENTS]
6.   - function prototype:
7.   - initial_map:
8.   - shared_info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
9.   - name: 0x00f3196e1c09 <String[9]: #addstring>
10.   - formal_parameter_count: 2
11.   - kind: NormalFunction
12.   - context: 0x00f3196c34a9 <NativeContext[262]>
13.   - code: 0x021668583001 <Code TURBOFAN>
14.   - interpreted
15.   - bytecode: 0x00f3196e20c1 <BytecodeArray[6]>
16.   - source code: (a,b) {
17.       return a+b;
18.  }
19.   - properties: 0x03e492c81309 <FixedArray[0]>
20.   - feedback vector: 000000F3196E2119: [FeedbackVector] in OldSpace
21.   - map: 0x03e492c81b69 <Map>
22.   - length: 1
23.   - shared function info: 0x00f3196e1db1 <SharedFunctionInfo addstring>
24.   - no optimized code
25.   - optimization marker: OptimizationMarker::kNone
26.   - optimization tier: OptimizationTier::kNone
27.   - invocation count: 0
28.   - profiler ticks: 0
29.   - closure feedback cell array: 000003E492C82FB9: [ClosureFeedbackCellArray]
in ReadOnlySpace
30.   - map: 0x03e492c81f19 <Map>
31.   - length: 0
32.   - slot #0 BinaryOp BinaryOp:String {
33.       [0]: 16
34.    }
35.  000000AE91D413A1: [Map]
36.   - type: JS_FUNCTION_TYPE
37.   - instance size: 64
38.   - inobject properties: 0
39.   - elements kind: HOLEY_ELEMENTS
40.   - unused property fields: 0
41.   - enum length: invalid
42.   - stable_map
43.   - callable
44.   - constructor
45.   - has_prototype_slot
46.   - back pointer: 0x03e492c81599 <undefined>
```

```
47.    - prototype_validity cell: 0x01cfdce84a09 <Cell value= 1>
48.    - instance descriptors (own) #5: 0x00f3196c4041 <DescriptorArray[5]>
49.    - prototype: 0x00f3196c3ee9 <JSFunction (sfi = 000001CFDCE91269)>
50.    - constructor: 0x00f3196c3fd9 <JSFunction Function (sfi = 000001CFDCE91409)>
51.    - dependent code: 0x03e492c81239 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
52.    - construction counter: 0
```

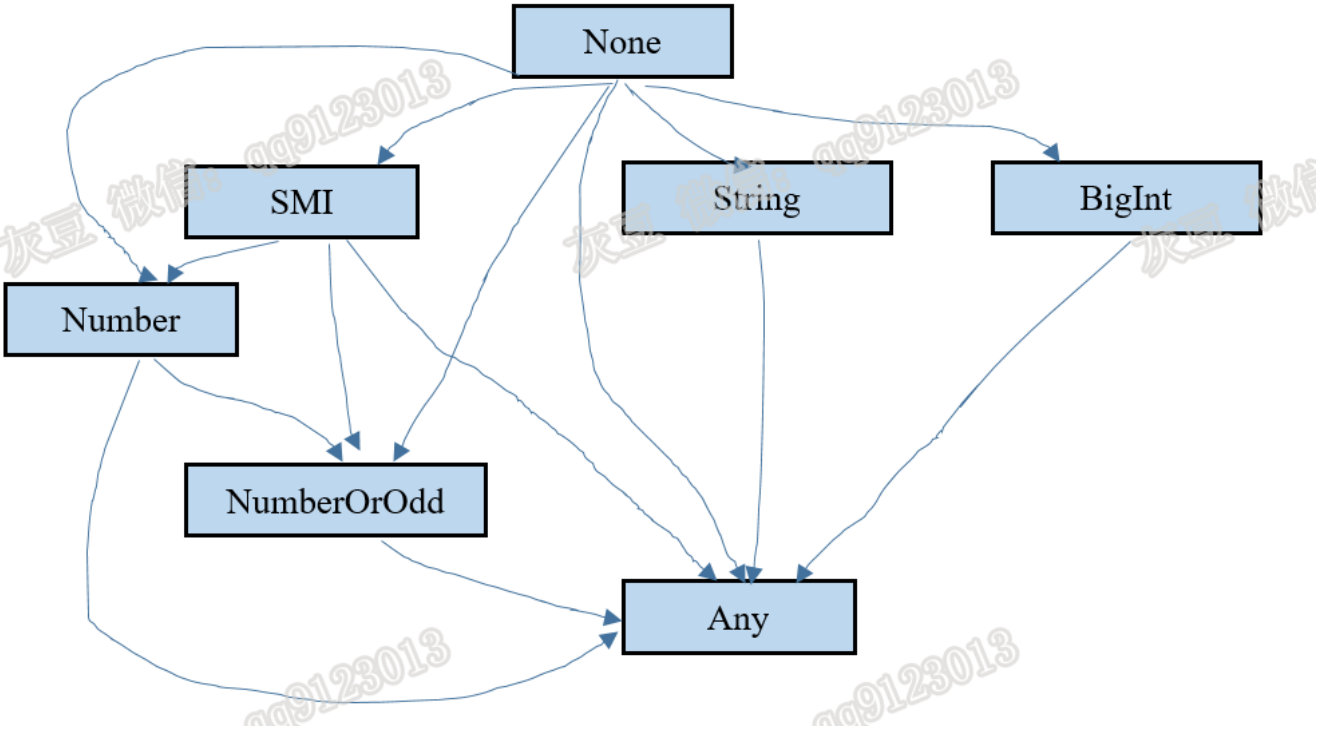
上述信息是执行 Turbofan 后的 debug 输出。第 1 行是输出结果；与执行 Turbofan 前的 debug 输出的区别是：

(1) 第 13 行信息说明运行方式为 Code TURBOFAN ，而不是 InterpreterEntryTrampoline。

(2) 第 32 行信息说明当前 feedback vector slot 是对字符串的操作，也就是为字符串操作做优化。

feedback 是一种收集运行时信息的机制，之所以叫 vector，是因为他是用 C++ 的 vector 实现的。feedback 可以用于指导 inline cache 存储相关信息，也可以用于指导 Turbofan 工作。根据 JavaScript 的数据类型，我们可能粗略推算出 feedback 的状态如图 2所示。

Figure 2



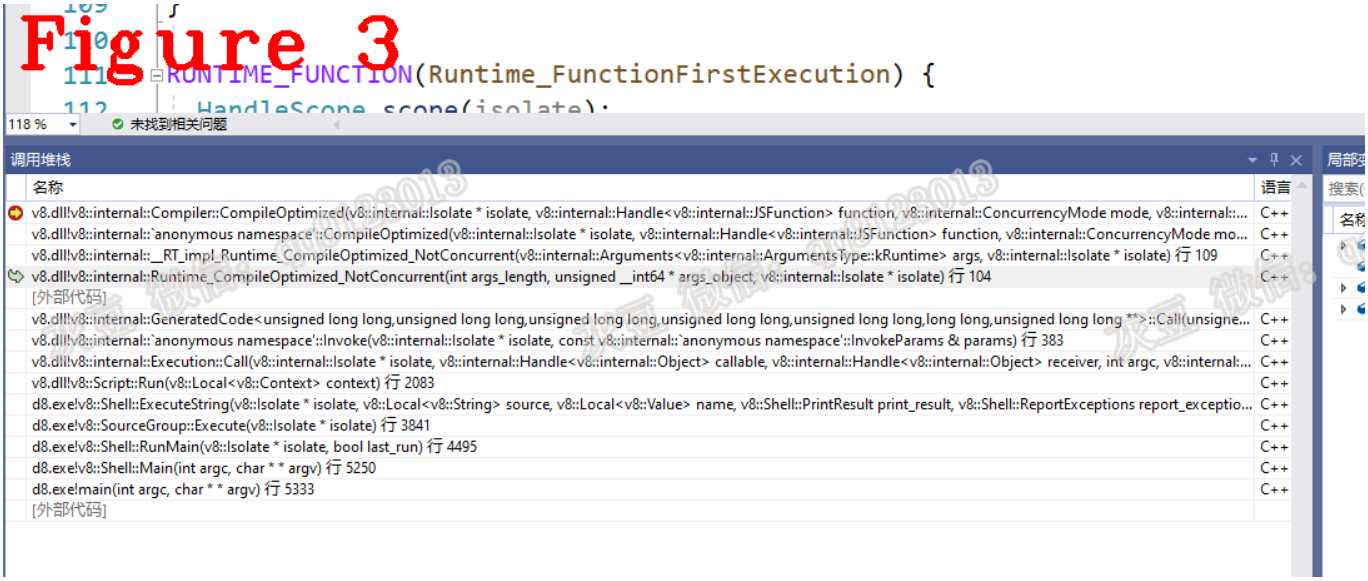
V8 开始执行时 feedback 是 None，因为此时我们什么也没有，也看不到任何信息。由于 addstring 常用于字符串相加，所以运行一段时间后 feedback 变成了 String，这时就可以使用 Turbofan 对字符串加法做优化。随着程序的运行 feedback 又变成了 Any，这时意味着 addstring 常被用于任何数据的相加。**注意** feedback 的状态一般是自上而下转换的，如果逆向转换就可能发生去优化。

本文使用 %OptimizeFunctionOnNextCall() 主动唤起 Turbofan，源码如下：

```
1.  RUNTIME_FUNCTION(Runtime_OptimizeFunctionOnNextCall) {
2.    HandleScope scope(isolate);
3.    return OptimizeFunctionOnNextCall(args, isolate,
TierupKind::kTierupBytecode);
4.  }
5.  //.................分隔线.............
6.  bool Compiler::CompileOptimized(Isolate* isolate, Handle<JSFunction> function,
7.                                  ConcurrencyMode mode, CodeKind code_kind) {
```

```
8.    DCHECK(CodeKindIsOptimizedJSFunction(code_kind));
9.    DCHECK(AllowCompilation::IsAllowed(isolate));
10.    if (FLAG_stress_concurrent_inlining &&
11.        isolate->concurrent_recompilation_enabled() &&
12.        mode == ConcurrencyMode::kNotConcurrent &&
13.        isolate->node_observer() == nullptr) {
14.      SpawnDuplicateConcurrentJobForStressTesting(isolate, function, mode,
15.                                                  code_kind);
16.    }
17.    Handle<Code> code;
18.    if (!GetOptimizedCode(isolate, function, mode, code_kind).ToHandle(&code))
{
19.      // Optimization failed, get the existing code. We could have optimized
code
20.      // from a lower tier here. Unoptimized code must exist already if we are
21.      // optimizing.
22.      DCHECK(!isolate->has_pending_exception());
23.      DCHECK(function->shared().is_compiled());
24.      DCHECK(function->shared().IsInterpreted());
25.      code = ContinuationForConcurrentOptimization(isolate, function);
26.    }
27.    function->set_code(*code, kReleaseStore);
28.    // Check postconditions on success.
29.    return true;
30.  }
```

上述是两部分代码：Runtime_OptimizeFunctionOnNextCall 设置 Turbofan 的使能标记为真，设置使能标记后会调用 Runtime_CompileOptimized_NotConcurrent 方法，在该方法中调用上述代码中的 CompileOptimized。图 3 给出了 CompileOptimized 方法的调用堆栈。



技术总结

（1） 只有使用 --allow-natives-syntax 选项时，才能在 JavaScript 中使用 native 指令；

（2） feedback 的状态发生逆向转换时可能导致去优化。

好了，今天到这里，下次见。

个人能力有限，有不足与纰漏，欢迎批评指正

微信：qq9123013 备注：v8交流 知乎：https://www.zhihu.com/people/v8blink

本文由灰豆原创发布
转载出处： https://www.anquanke.com/post/id/262579
安全客 - 有思想的安全新媒体