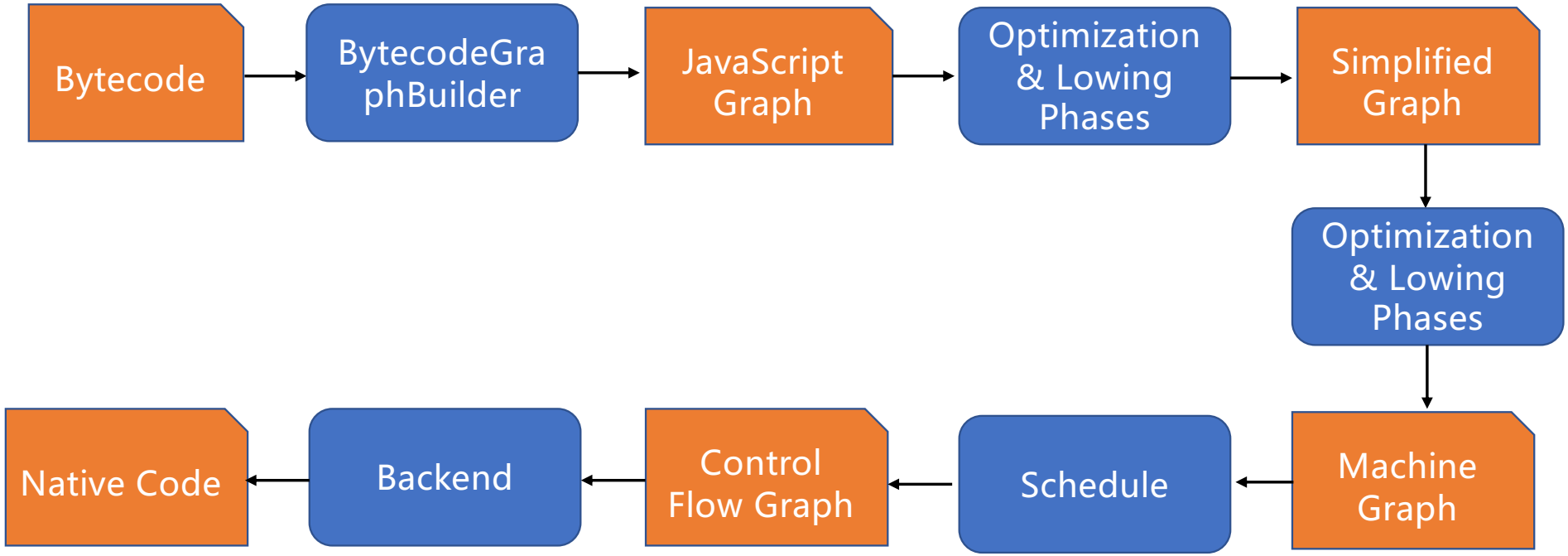
ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

V8 Turbofan:
从字节码到SON图-part1

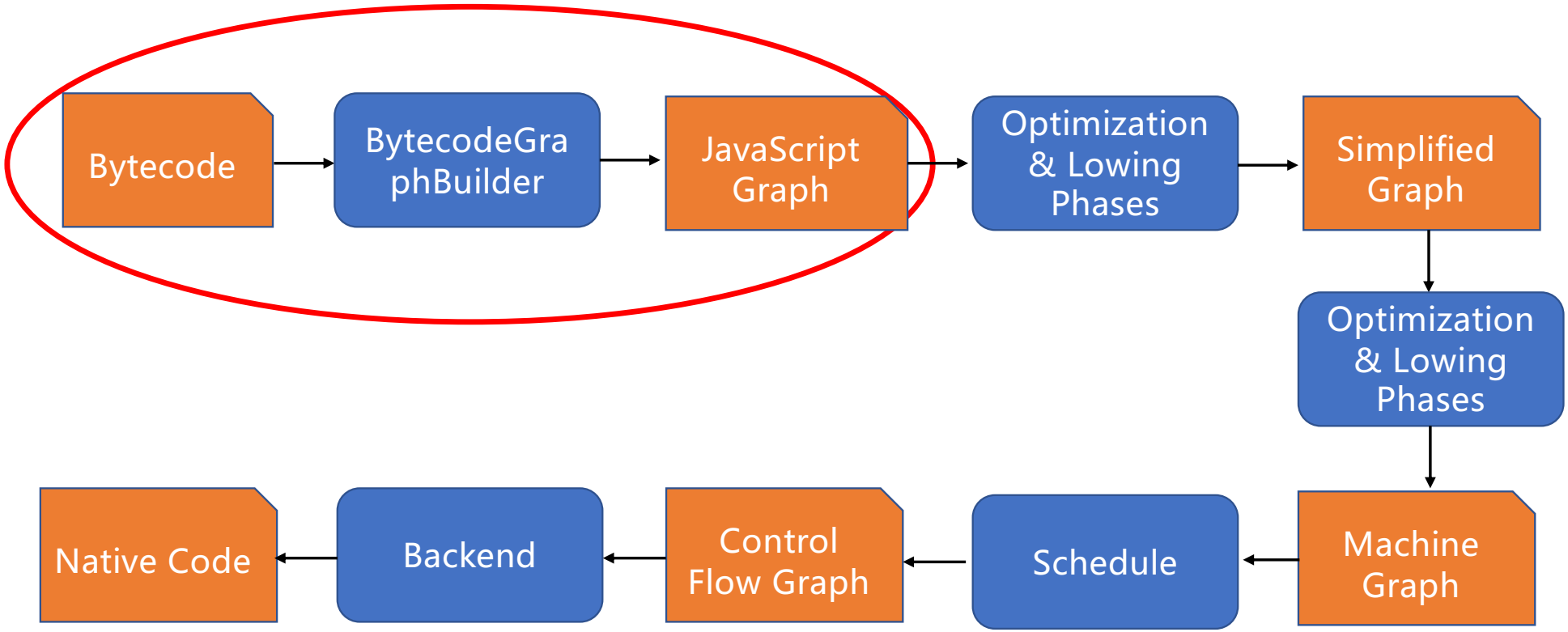
PLCT实验室 邱吉
qiuji@iscas.ac.cn

2022/03/09

上一节课的回顾：TurboFan的Pipeline



接下来，以一个只有三个字节码的简单例子，来说明SON图构建过程



从Bytecode到SON图的构建：内容大纲

- Demo case 及其 Graph
- 总体构建流程概述
- Step by Step, Node by Node讲述图的构建过程
    1. new and set the Start node
    2. new Environment and set_environment
    3. CreateFeedbackCellNode
    4. CreateFeedbackVectorNode
    5. MaybeBuildTierUpCheck
    6. CreateNativeContextNode
    7. VisitBytecodes: one by one
    8. new and set the End node

从Bytecode到SON图的构建：内容大纲

- Demo case 及其 Graph
- 总体构建流程概述
- Step by Step, Node by Node讲述图的构建过程
    1. new and set the Start node
    2. new Environment and set_environment
    3. CreateFeedbackCellNode
    4. CreateFeedbackVectorNode
    5. MaybeBuildTierUpCheck
    6. CreateNativeContextNode
    7. VisitBytecodes: one by one
    8. new and set the End node

Demo case: add.js

```
function test(v) {
return v+1;
}
%PrepareFunctionForOptimization(test);
test(1);
%OptimizeFunctionOnNextCall(test);
test(2);
```

生成用于turbolizer的json文件的command line:
./d8 --trace-turbo --allow-natives-syntax add.js
运行目录下生成turbo-test-0.json文件，可以通过turbolizer加载

```
[generated bytecode for function: test
(0x007f04662411 <SharedFunctionInfo test>)]
Bytecode length: 6
Parameter count 2
Register count 0
Frame size 0
OSR nesting level: 0
Bytecode Age: 0
   19 S> 0x7f04662766 @    0 : 0b 03           Ldar a0
   27 E> 0x7f04662768 @    2 : 44 01 00        AddSmi [1], [0]
   30 S> 0x7f0466276b @    5 : a8              Return
Constant pool (size = 0)
Handler Table (size = 0)
Source Position Table (size = 8)
```

生成文本输出的command line:
./d8 --trace-turbo-graph --print-bytecode --allow-natives-syntax add.js 2>&1 | tee log.txt

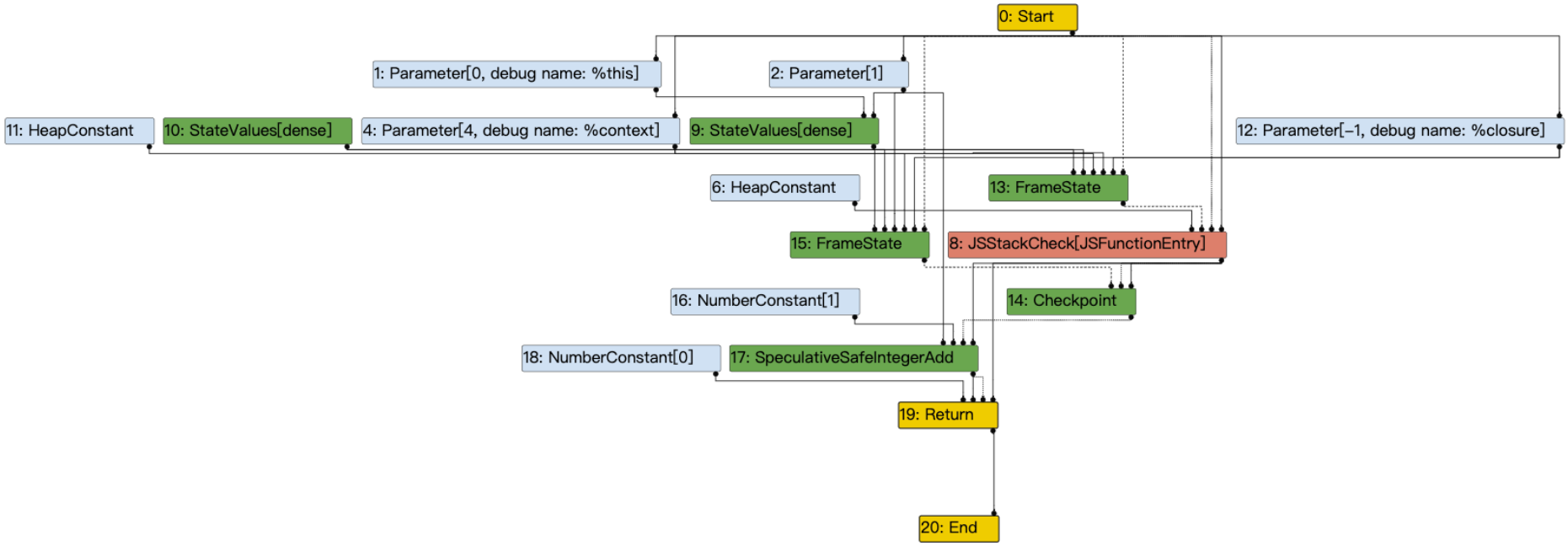
log.txt与SON构建相关的部分内容

Concurrent recompilation has been disabled for tracing.

Begin compiling method test using TurboFan
-- Graph after V8.TFBytecodeGraphBuilder --
#18:NumberConstant[0]()
#0:Start()
#2:Parameter[1](#0:Start)
#16:NumberConstant[1]()
#1:Parameter[0, debug name: %this](#0:Start)
#9:StateValues[dense](#1:Parameter, #2:Parameter)
#10:StateValues[dense]()
#4:Parameter[4, debug name: %context](#0:Start)
#12:Parameter[-1, debug name: %closure](#0:Start)
#15:FrameState[UNOPTIMIZED_FRAME, 2, Ignore, 0x00fe9a322411 <SharedFunctionInfo test>](#9:StateValues, #10:StateValues, #2:Parameter, #4:Parameter, #12:Parameter, #0:Start)
#6:HeapConstant[0x00fe9a3038c9 <NativeContext[252]>]()
#11:HeapConstant[0x007fb5ec1de9 <Odd Oddball: optimized_out>]()
#13:FrameState[UNOPTIMIZED_FRAME, -1, Ignore, 0x00fe9a322411 <SharedFunctionInfo test>](#9:StateValues, #10:StateValues, #11:HeapConstant, #4:Parameter, #12:Parameter, #0:Start)
#8:JSStackCheck[JSFunctionEntry](#6:HeapConstant, #13:FrameState, #0:Start, #0:Start)
#14:Checkpoint(#15:FrameState, #8:JSStackCheck, #8:JSStackCheck)
#17:SpeculativeSafeIntegerAdd[SignedSmall](#2:Parameter, #16:NumberConstant, #14:Checkpoint, #8:JSStackCheck)
#19:Return(#18:NumberConstant, #17:SpeculativeSafeIntegerAdd, #17:SpeculativeSafeIntegerAdd, #8:JSStackCheck)
#20:End(#19:Return)

红色：节点编号
绿色：节点的操作符名称
黄色：节点的属性
灰色：节点的输入

Turbolize的可视化展示



https://v8.github.io/tools/head/turbolizer/index.html，按照使用说明加载turbo-test-0.json后生成的图

从Bytecode到SON图的构建：内容大纲

- Demo case 及其 Graph
- 总体构建流程概述
- Step by Step, Node by Node讲述图的构建过程
    1. new and set the Start node
    2. new Environment and set_environment
    3. CreateFeedbackCellNode
    4. CreateFeedbackVectorNode
    5. MaybeBuildTierUpCheck
    6. CreateNativeContextNode
    7. VisitBytecodes: one by one
    8. new and set the End node

总体构建流程概述

- PipelineCompilationJob::ExecuteJobImpl: 调用JS函数编译Job
- PipelineImpl::CreateGraph(): CreateGraph的入口
- Run<GraphBuilderPhase>: 调用GraphBuilderPhase
- BuildGraphFromBytecode(): 调用BuildGraphFromBytecode函数，new BytecodeGraphBuilder对象
- BytecodeGraphBuilder::CreateGraph: 调用CreateGraph成员函数，逐个建立节点
    1. new and set start node
    2. new Environment and set_environment
    3. CreateFeedbackCellNode
    4. CreateFeedbackVectorNode
    5. MaybeBuildTierUpCheck
    6. CreateNativeContextNode
    7. VisitBytecodes: one by one
    8. new and set end node

从Bytecode到SON图的构建：内容大纲

- Demo case 及其 Graph
- 总体构建流程概述
- Step by Step, Node by Node讲述图的构建过程
    1. new and set the Start node
    2. new Environment and set_environment
    3. CreateFeedbackCellNode
    4. CreateFeedbackVectorNode
    5. MaybeBuildTierUpCheck
    6. CreateNativeContextNode
    7. VisitBytecodes: one by one
    8. new and set the End node

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step1: new and set the Start Node

@src/compiler/bytecode-graph-builder.cc: void BytecodeGraphBuilder::CreateGraph()

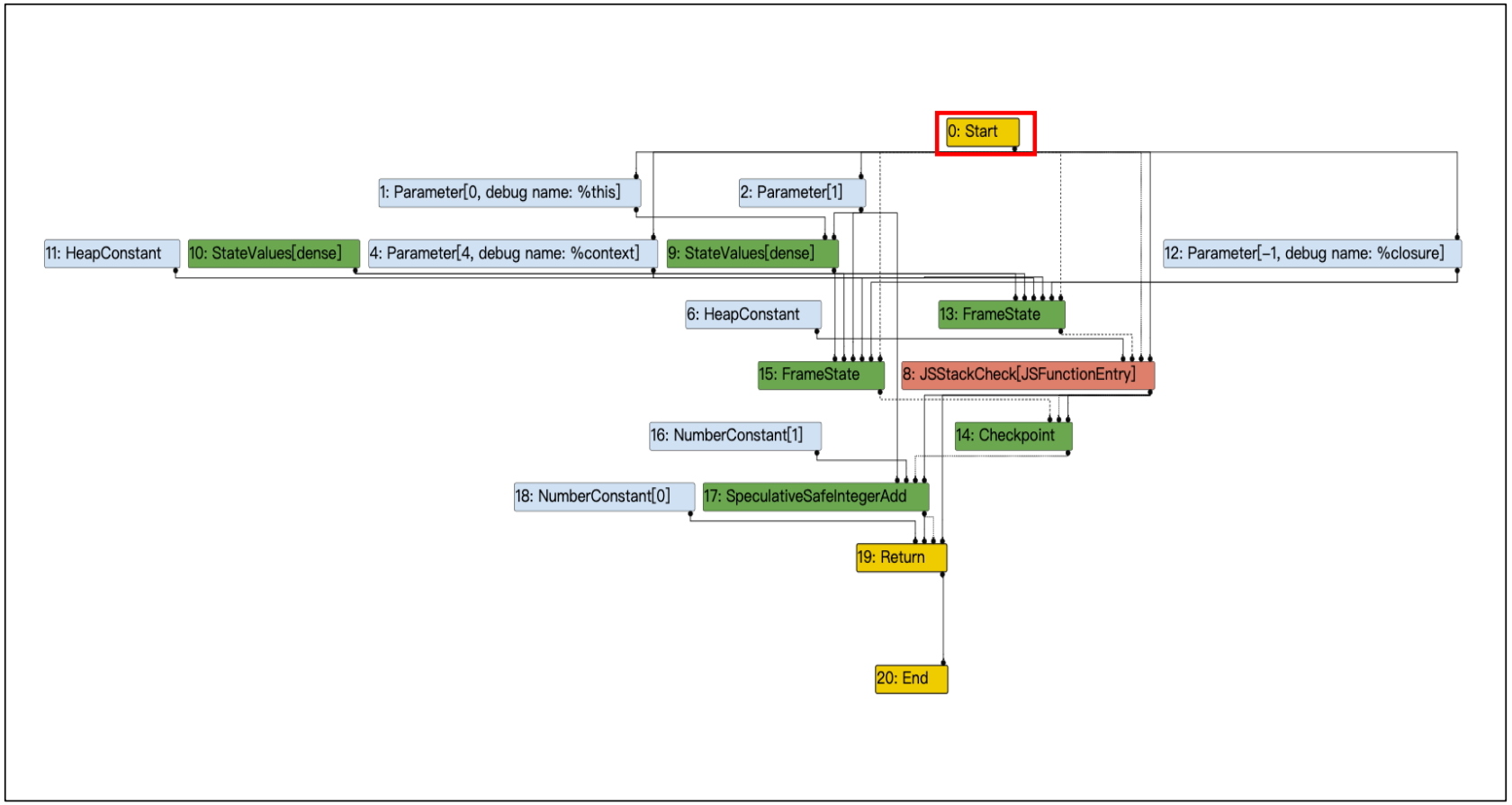
```
int start_output_arity = StartNode::OutputArityForFormalParameterCount(
      bytecode_array().parameter_count()); //Arity是操作数的意思，start节点的value output是所有的parameter
      graph()->SetStart(graph()->NewNode(common()->Start(start_output_arity)));
/*  代码语句说明
1.   通过graph()得到Graph类型的图对象：graph()调用JSGraph类型对象成员jsgraph的graph()函数, 因为JSGraph
公有继承MachineGraph，所以最终调用到MachineGraph的graph()函数。

2.  JSGraph包装了SON图的数据Graph，及其变换所需的
Common/JS/Simplified/Machine Operator
Builder，这些Builder用于在创建图中的各类节点。*/
```

```
  JSGraph(Isolate* isolate, Graph* graph, CommonOperatorBuilder* common,
          JSOperatorBuilder* javascript, SimplifiedOperatorBuilder* simplified,
          MachineOperatorBuilder* machine)
      : MachineGraph(graph, common, machine), isolate_(isolate), javascript_(javascript), simplified_(simplified) {
  }
```

```
Graph::Graph(Zone* zone)
    : zone_(zone), start_(nullptr), end_(nullptr),  mark_max_(0), next_node_id_(0), decorators_(zone) {
  CHECK_IMPLIES(kCompressGraphZone, zone->supports_compression());
}
```

已构建完成的节点


0: Start
1: Parameter[0, debug name: %this]
2: Parameter[1]
11: HeapConstant
10: StateValues[dense]
4: Parameter[4, debug name: %context]
9: StateValues[dense]
12: Parameter[-1, debug name: %closure]
6: HeapConstant
13: FrameState
15: FrameState
8: JSStackCheck[JSFunctionEntry]
16: NumberConstant[1]
14: Checkpoint
18: NumberConstant[0]
17: SpeculativeSafeIntegerAdd
19: Return
20: End

节点列表

0	Start	

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step2.1: new Environment and set_environment

@src/compiler/bytecode-graph-builder.cc: void BytecodeGraphBuilder::CreateGraph()

```
Environment env(this, bytecode_array().register_count(),
                bytecode_array().parameter_count(),
                bytecode_array().incoming_new_target_or_generator_register(),
                graph()->start());
  set_environment(&env);
```

1. SON图是按照字节码的顺序逐条构建的
2. Environment模拟了解释器在执行每一条字节码时所对应的上下文状态。字节码通过env来读取或者修改解释器上下文

Environment Class的部分成员：

```
BytecodeGraphBuilder* builder_;
  int register_count_; //Bytecode对应的解释器寄存器数量
  int parameter_count_;//参数数量
  Node* context_;//上下文
  Node* control_dependency_;
  Node* effect_dependency_;
  NodeVector values_;//Node数组，用于保存重要的上下文节点
  Node* parameters_state_values_;
  Node* generator_state_;
  int register_base_;//values_中寄存器的基索引
  int accumulator_base_;//values_中累加器的索引
```

Step2.2: new Environment and set_environment

@src/compiler/bytecode-graph-builder.cc: void BytecodeGraphBuilder::CreateGraph()

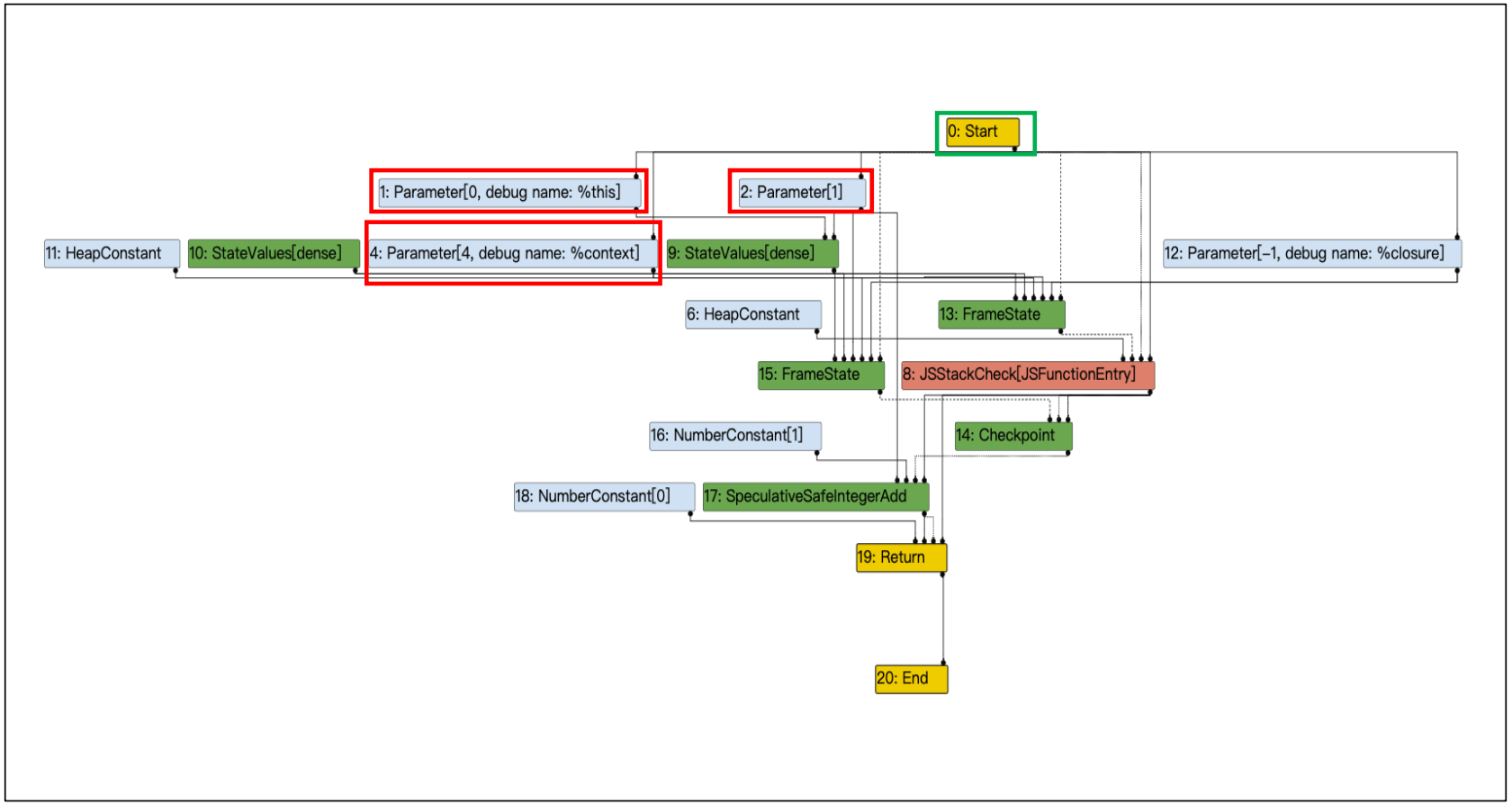
```
// Parameters including the receiver
  for (int i = 0; i < parameter_count; i++) {
    const char* debug_name = (i == 0) ? "%this" : nullptr;
    Node* parameter = builder->GetParameter(i, debug_name);
    values()->push_back(parameter);
  }
  // Registers
  register_base_ = static_cast<int>(values()->size());
  Node* undefined_constant = builder->jsgraph()->UndefinedConstant();
  values()->insert(values()->end(), register_count, undefined_constant);
  // Accumulator
  accumulator_base_ = static_cast<int>(values()->size());
  values()->push_back(undefined_constant);
  // Context
  int context_index = Linkage::GetJSCallContextParamIndex(parameter_count);
  context_ = builder->GetParameter(context_index, "%context");
```

3. Environment的构造函数，会将Start节点设置成初始的control_dependency和effect_dependency
4. 最重要的一步是进行values_节点向量的构建，该向量的layout如下：

   [receiver][parameters][registers][accumulator]

   其中，Node receiver和Node parameter 来自于cached_parameters_数组。在env构造时分配register和accumulator的值，会在SON图构建过程中进行绑定，在env构造时，会先把所有register的值只向一个UndefinedConstant节点。

已构建完成的节点


0: Start
1: Parameter[0, debug name: %this]
2: Parameter[1]
11: HeapConstant
10: StateValues[dense]
4: Parameter[4, debug name: %context]
9: StateValues[dense]
12: Parameter[-1, debug name: %closure]
6: HeapConstant
13: FrameState
15: FrameState
8: JSStackCheck[JSFunctionEntry]
16: NumberConstant[1]
14: Checkpoint
18: NumberConstant[0]
17: SpeculativeSafeIntegerAdd
19: Return
20: End

节点列表

0	Start	
1	Parameter	*this
2	Parameter	1
3	HeapConstant	UndefinedConstant（用于values_向量中的register和accumulator初始值）
4	Parameter	context

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step3：CreateFeedbackCellNode

@src/compiler/bytecode-graph-builder.cc

```
void BytecodeGraphBuilder::CreateFeedbackCellNode() {
  DCHECK_NULL(feedback_cell_node_);
  // Only used by tier-up logic; for code that doesn't tier-up, we can skip this.
  if (!CodeKindCanTierUp(code_kind())) return; // 如果当前的CodeKind不再可以升级，就不创建
FeedbackCellNode了，这意味着只有在TurboProp编译的时候，才会产生FeedbackCellNode
  feedback_cell_node_ = jsgraph()->Constant(feedback_cell_);
}
```

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step4：CreateFeedbackVectorNode

@src/compiler/bytecode-graph-builder.cc

```
void BytecodeGraphBuilder::CreateFeedbackVectorNode() {
  DCHECK_NULL(feedback_vector_node_);
  feedback_vector_node_ = jsgraph()->Constant(feedback_vector());
}
//在SON图中会有一个专门的用于表示当前feedback vector的节点，它在开始处理字节码之前生成，它是一个常量节点
```

节点列表

0	Start	
1	Parameter	*this
2	Parameter	1
3	HeapConstant	UndefinedConstant（用于values_向量中的register和accumulator初始值）
4	Parameter	context
5	HeapConstant	指向FeedBackVector

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step5：MaybeBuildTierUpCheck

@src/compiler/bytecode-graph-builder.cc

```
void BytecodeGraphBuilder::MaybeBuildTierUpCheck() {
  if (!CodeKindCanTierUp(code_kind()) || skip_tierup_check()) return; //如果code不会升级，就不生成Check节点了
  int parameter_count = bytecode_array().parameter_count();
  Node* target = GetFunctionClosure();
  Node* new_target = GetParameter(
      Linkage::GetJSCallNewTargetParamIndex(parameter_count), "%new.target");
  Node* argc = GetParameter(
      Linkage::GetJSCallArgCountParamIndex(parameter_count), "%argc");
  DCHECK_EQ(environment()->Context()->opcode(), IrOpcode::kParameter);
  Node* context = environment()->Context();
  NewNode(simplified()->TierUpCheck(), feedback_vector_node(), target,
          new_target, argc, context); //否则就直接生成Simplified TierUpCheck节点
}
```

Step by Step, Node by Node讲述图的构建过程

1. new and set the Start node
2. new Environment and set_environment
3. CreateFeedbackCellNode
4. CreateFeedbackVectorNode
5. MaybeBuildTierUpCheck
6. CreateNativeContextNode
7. VisitBytecodes: one by one
8. new and set the End node

Step6 : CreateNativeContextNode

@src/compiler/bytecode-graph-builder.cc

```
void BytecodeGraphBuilder::CreateNativeContextNode() {
  DCHECK_NULL(native_context_node_);
  native_context_node_ = jsgraph()->Constant(native_context()); //生成Context节点
}
```

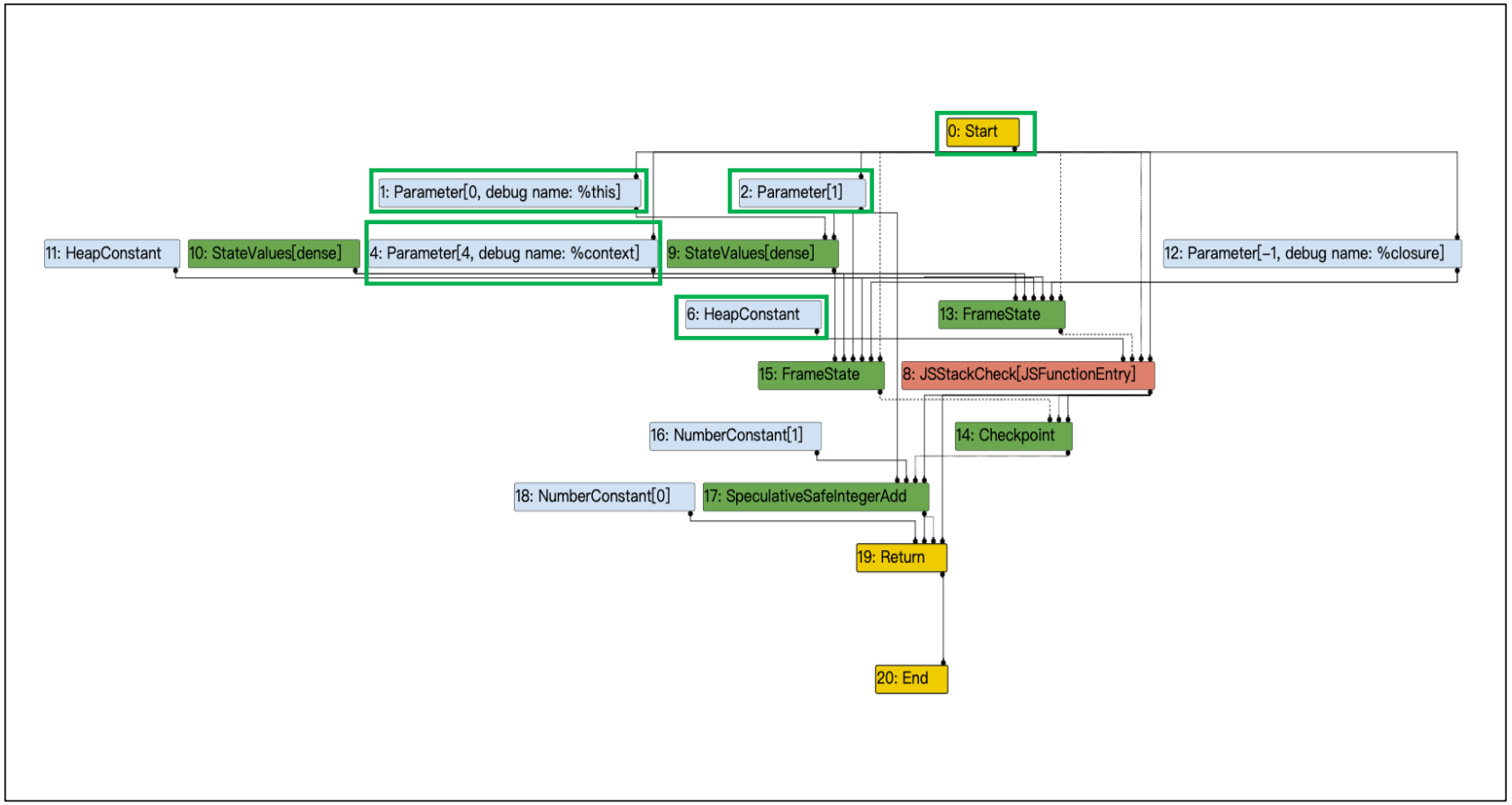
节点列表

0	Start	
1	Parameter	*this
2	Parameter	1
3	HeapConstant	UndefinedConstant（用于values_向量中的register和accumulator初始值）
4	Parameter	context
5	HeapConstant	指向FeedBackVector
6	HeapConstant	nativecontext

本节课内容总结：从Bytecode到SON图的构建Part1

- Demo case 及其 Graph
- 总体构建流程概述
- Step by Step, Node by Node讲述图的构建过程
  1. new and set the Start node
  2. new Environment and set_environment
  3. CreateFeedbackCellNode
  4. CreateFeedbackVectorNode
  5. MaybeBuildTierUpCheck
  6. CreateNativeContextNode
  7. VisitBytecodes: one by one
  8. new and set the End node

已构建完成的节点


0: Start
1: Parameter[0, debug name: %this]
2: Parameter[1]
11: HeapConstant
10: StateValues[dense]
4: Parameter[4, debug name: %context]
9: StateValues[dense]
12: Parameter[-1, debug name: %closure]
6: HeapConstant
13: FrameState
15: FrameState
8: JSStackCheck[JSFunctionEntry]
16: NumberConstant[1]
14: Checkpoint
18: NumberConstant[0]
17: SpeculativeSafeIntegerAdd
19: Return
20: End

谢 谢

欢迎交流合作