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

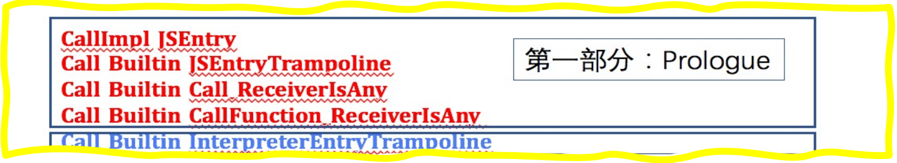
V8中HelloWorld的解释执行过程-part4

智能软件研究中心 邱吉
qiuji@iscas.ac.cn

2021/11/12

前情回顾

- part1：
  - hello.js：print(“HelloWorld!”)的字节码和含义
  - 如何从--trace-sim的log文件中，梳理hello.js的解释执行过程
- part2：
  - d8上hello.js的整体执行流程
  - 如何进入第一部分Prologue部分开始执行
- part3：
  - Ignition解释循环主体 InterpreterEntryTrampoline的整体控制流程



```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin InterpreterEntryTrampoline
```

第一部分：Prologue

Hello.js step by step- 本次内容

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器执行主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

今天的内容：

1. Ignition 是如何逐个执行BytecodeHandler的

2. 解释结束后如何返回

本次内容

- Ignition如何逐个执行BytecodeHandler
- 所有的字节码解释执行完成后如何返回
- 调试的代码和log： https://github.com/qjivy/v8/tree/v8ignition-learn

复习：hello.js的字节码

```
./d8 --print-bytecode hello.js
```

```
print(” hello” )
```

```
Bytecode length: 13
Parameter count 1
Register count 3
Frame size 24
OSR nesting level: 0
Bytecode Age: 0
        0xdf23ca206e @    0 : 21 00 00        LdaGlobal [0], [0]
        0xdf23ca2071 @    3 : c2              Star1
        0xdf23ca2072 @    4 : 13 01           LdaConstant [1]
        0xdf23ca2074 @    6 : c1              Star2
        0xdf23ca2075 @    7 : 61 f9 f8 02     CallUndefinedReceiver1 r1, r2, [2]
        0xdf23ca2079 @   11 : c3              Star0
        0xdf23ca207a @   12 : a8              Return
Constant pool (size = 2)
0xdf23ca2019: [FixedArray] in OldSpace
 - map: 0x001bf11012c1 <Map>
 - length: 2
          0: 0x00df23c813a9 <String[5]: #print>
          1: 0x00df23ca1f71 <String[5]: #hello>
Handler Table (size = 0)
Source Position Table (size = 0)
```

复习：hello.js的字节码

1	LdaGlobal [0], [0]	LdaGlobal <name_index> <slot> Load the global with name in constant pool entry <name_index> into the accumulator using FeedBackVector slot <slot>.
2	Star1	Store the accumulator into r1.
3	LdaConstant [1]	LdaConstant <idx> Load constant literal at |idx| in the constant pool into the accumulator.
4	Star2	Store the accumulator into r2.
5	CallUndefinedReceiver1 r1, r2, [2]	Call <callable> <receiver> <arg_count> <feedback_slot_id> Call a JSfunction or Callable in |callable| with the |receiver| and |arg_count| arguments in subsequent registers. Collect type feedback into |feedback_slot_id|
6	Star0	Store the accumulator into r0.
7	Return	Return the value in the accumulator.

- 字节码的含义和格式，可以参考src/interpreter/interpreter-generator.cc 文件
- 蓝色的部分是feedback vector slot的索引号，用于字节码执行时类型信息的记录。如果只讨论解释器的执行流程，可以忽略掉它们

复习：概览./d8 --trace-sim hello.js 2>&1 |tee logtracesim.txt

- grep搜索所有的“Call to” 和 “Return to” 的行，就可以得到执行流如何在各个builtins中传递
- 蓝色的部分就是解释器Ignition执行过程

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器执行主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

Bytecode-1: LdaGlobal

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

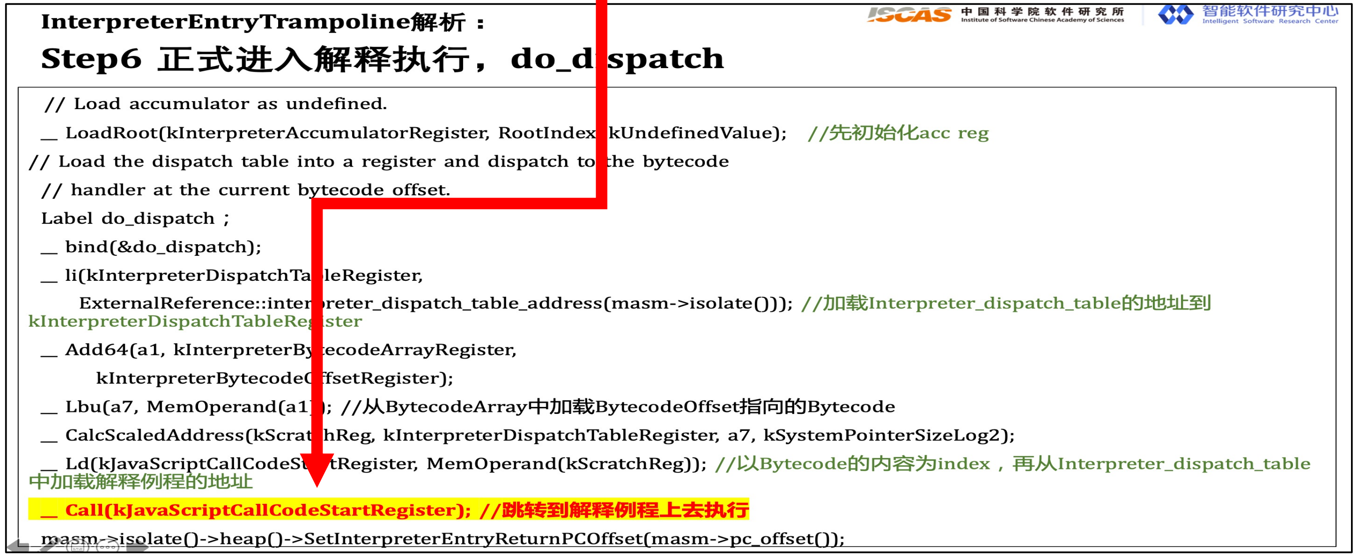
```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

Bytecode-1: 分发到LdaGlobalHandler

- InterpreterEntryTrampoline从Call(kJavaScriptCallCodeStartRegister)跳转到LdaGlobalHandler的起始地址执行

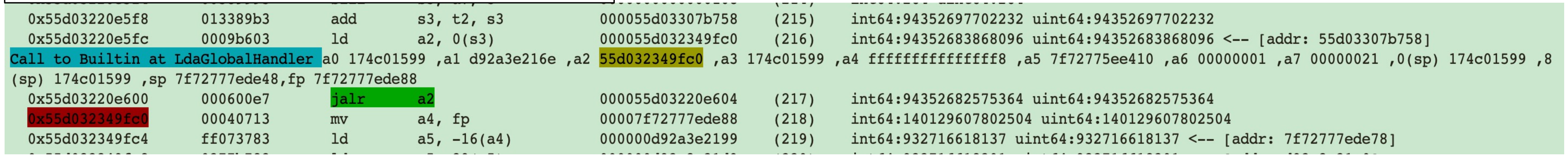


InterpreterEntryTrampoline解析：

Step6 正式进入解释执行， do_dispatch

```
 // Load accumulator as undefined.
 __ LoadRoot(kInterpreterAccumulatorRegister, RootIndex::kUndefinedValue);   //先初始化acc reg
// Load the dispatch table into a register and dispatch to the bytecode
 // handler at the current bytecode offset.
 Label do_dispatch ;
 __ bind(&do_dispatch);
 __ li(kInterpreterDispatchTableRegister,
      ExternalReference::interpreter_dispatch_table_address(masm->isolate())); //加载Interpreter_dispatch_table的地址到
kInterpreterDispatchTableRegister
 __ Add64(a1, kInterpreterBytecodeArrayRegister,
        kInterpreterBytecodeOffsetRegister);
 __ Lbu(a7, MemOperand(a1)); //从BytecodeArray中加载BytecodeOffset指向的Bytecode
 __ CalcScaledAddress(kScratchReg, kInterpreterDispatchTableRegister, a7, kSystemPointerSizeLog2);
 __ Ld(kJavaScriptCallCodeStartRegister, MemOperand(kScratchReg)); //以Bytecode的内容为index，再从Interpreter_dispatch_table
中加载解释例程的地址
 __ Call(kJavaScriptCallCodeStartRegister); //跳转到解释例程上去执行
 masm->isolate()->heap()->SetInterpreterEntryReturnPCOffset(masm->pc_offset());
```

```
./d8 --trace-sim print.js >log.txt
```



```
  0x55d03220e5f8      013389b3       add       s3, t2, s3          000055d03307b758    (215)    int64:94352697702232 uint64:94352697702232
  0x55d03220e5fc      0009b603       ld        a2, 0(s3)           000055d032349fc0    (216)    int64:94352683868096 uint64:94352683868096 <-- [addr: 55d03307b758]
Call to Builtin at LdaGlobalHandler a0 174c01599 ,a1 d92a3e216e ,a2 55d032349fc0 ,a3 174c01599 ,a4 fffffffffffffff8 ,a5 7f72775ee410 ,a6 00000001 ,a7 00000021 ,0(sp) 174c01599 ,8
(sp) 174c01599 ,sp 7f72777ede48,fp 7f72777ede88
  0x55d03220e600      000600e7       jalr      a2                  000055d03220e604    (217)    int64:94352682575364 uint64:94352682575364
  0x55d032349fc0      00040713       mv        a4, fp              00007f72777ede88    (218)    int64:140129607802504 uint64:140129607802504
  0x55d032349fc4      ff073783       ld        a5, -16(a4)         000000d92a3e2199    (219)    int64:932716618137 uint64:932716618137 <-- [addr: 7f72777ede78]
```

Bytecode-1: LdaGlobalHandler的生成函数1

LdaGlobal [0], [0]	LdaGlobal <name_index> <slot> Load the global with name in constant pool entry <name_index> into the accumulator using FeedBackVector slot <slot>.

@ interpreter/interpreter-generator.cc

```
// LdaGlobal <name_index> <slot>
//
// Load the global with name in constant pool entry <name_index> into the
// accumulator using FeedBackVector slot <slot> outside of a typeof.
IGNITION_HANDLER(LdaGlobal, InterpreterLoadGlobalAssembler) {
  static const int kNameOperandIndex = 0;
  static const int kSlotOperandIndex = 1;
  LdaGlobal(kSlotOperandIndex, kNameOperandIndex, TypeofMode::kNotInside);
}
```

Bytecode-1: LdaGlobalHandler的生成函数2

@interpreter/interpreter-generator.cc

```
void LdaGlobal(int slot_operand_index, int name_operand_index,
                TypeofMode typeof_mode) {
  TNode<HeapObject> maybe_feedback_vector = LoadFeedbackVector();
  AccessorAssembler accessor_asm(state());
  ExitPoint exit_point(this, [=](TNode<Object> result) {
    SetAccumulator(result);
    Dispatch();
  });
  LazyNode<TaggedIndex> lazy_slot = [=] {
    return BytecodeOperandIdxTaggedIndex(slot_operand_index);
  };
  LazyNode<Context> lazy_context = [=] { return GetContext(); };
  LazyNode<Name> lazy_name = [=] {
    TNode<Name> name =
        CAST(LoadConstantPoolEntryAtOperandIndex(name_operand_index));
    return name;
  };
  accessor_asm.LoadGlobalIC(maybe_feedback_vector, lazy_slot, lazy_context,
                            lazy_name, typeof_mode, &exit_point);
 }
```

Bytecode-1: LdaGlobalHandle： 调用Builtin::kLoadGlobalIC_NoFeedback

@ic/accessor-assembler.cc

```
void AccessorAssembler::LoadGlobalIC(TNode<HeapObject> maybe_feedback_vector,
                                     const LazyNode<TaggedIndex>& lazy_slot,
                                     const LazyNode<Context>& lazy_context,
                                     const LazyNode<Name>& lazy_name,
                                     TypeofMode typeof_mode,
                                     ExitPoint* exit_point) {
  Label try_handler(this, Label::kDeferred), miss(this, Label::kDeferred),
      no_feedback(this, Label::kDeferred);

  GotoIf(IsUndefined(maybe_feedback_vector), &no_feedback);
  ...

  BIND(&no_feedback);
  {
    int ic_kind =
        static_cast<int>((typeof_mode == TypeofMode::kInside)
                             ? FeedbackSlotKind::kLoadGlobalInsideTypeof
                             : FeedbackSlotKind::kLoadGlobalNotInsideTypeof);
    exit_point->ReturnCallStub( //exit point将是Return的返回点
        Builtins::CallableFor(isolate(), Builtin::kLoadGlobalIC_NoFeedback), //继续调用Builtin
        lazy_context(), lazy_name(), SmiConstant(ic_kind));
  }
}
```

Bytecode-1: Builtin::kLoadGlobalIC_NoFeedback调用Builtin::kLoadIC_NoFeedback

@builtins/builtins-ic-gen.cc：

```
void Builtins::Generate_LoadGlobalIC_NoFeedback(
    compiler::CodeAssemblerState* state) {
  AccessorAssembler assembler(state);
  assembler.GenerateLoadGlobalIC_NoFeedback();
}
```

@ic/accessor-assembler.cc：

```
void AccessorAssembler::GenerateLoadGlobalIC_NoFeedback() {
  using Descriptor = LoadGlobalNoFeedbackDescriptor;
  auto name = Parameter<Object>(Descriptor::kName);
  auto context = Parameter<Context>(Descriptor::kContext);
  auto ic_kind = Parameter<Smi>(Descriptor::kICKind);
  LoadGlobalIC_NoFeedback(context, name, ic_kind);
}
```

@ic/accessor-assembler.cc：

```
void AccessorAssembler::LoadGlobalIC_NoFeedback(TNode<Context> context,
                                                TNode<Object> name,
                                                TNode<Smi> smi_typeof_mode) {
...
  TNode<JSGlobalObject> global_object =
      CAST(LoadContextElement(native_context, Context::EXTENSION_INDEX));
  TailCallStub(Builtins::CallableFor(isolate(), Builtin::kLoadIC_NoFeedback),
               context, global_object, name, smi_typeof_mode);
}
```

Bytecode-1: Builtin::kLoadIC_NoFeedback调用Runtime::kLoadNoFeedbackIC_Miss

@builtins/builtins-ic-gen.cc：

```
void Builtins::Generate_LoadIC_NoFeedback(compiler::CodeAssemblerState* state) {
  AccessorAssembler assembler(state);
  assembler.GenerateLoadIC_NoFeedback();
}
```

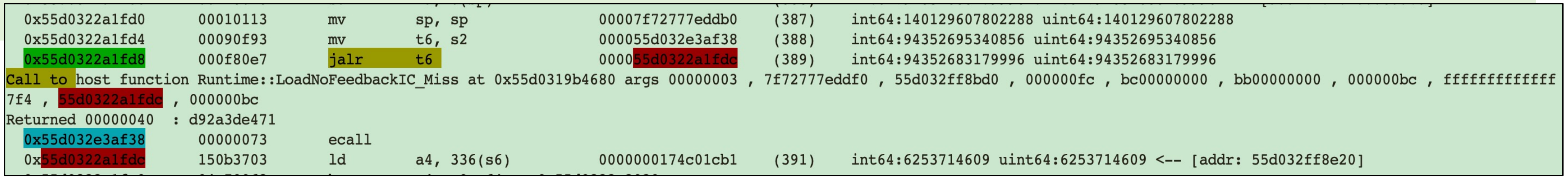
@ic/accessor-assembler.cc：

```
void AccessorAssembler::GenerateLoadIC_NoFeedback() {
  using Descriptor = LoadNoFeedbackDescriptor;
  auto receiver = Parameter<Object>(Descriptor::kReceiver);
  auto name = Parameter<Object>(Descriptor::kName);
  auto context = Parameter<Context>(Descriptor::kContext);
  auto ic_kind = Parameter<Smi>(Descriptor::kICKind);
  LoadICParameters p(context, receiver, name,
                     TaggedIndexConstant(FeedbackSlot::Invalid().ToInt()),
                     UndefinedConstant());
  LoadIC_NoFeedback(&p, ic_kind);
}
void AccessorAssembler::LoadIC_NoFeedback(const LoadICParameters* p,
                                          TNode<Smi> ic_kind) {
...
  BIND(&miss);
  {
    TailCallRuntime(Runtime::kLoadNoFeedbackIC_Miss, p->context(),
                    p->receiver(), p->name(), ic_kind);
  }}
```

Bytecode-1: Runtime::kLoadNoFeedbackIC_Miss函数

@ic/ic.cc:

```
RUNTIME_FUNCTION(Runtime_LoadNoFeedbackIC_Miss) {
  HandleScope scope(isolate);
  DCHECK_EQ(3, args.length());
  // Runtime functions don't follow the IC's calling convention.
  Handle<Object> receiver = args.at(0);
  Handle<Name> key = args.at<Name>(1);
  CONVERT_INT32_ARG_CHECKED(slot_kind, 2);
  FeedbackSlotKind kind = static_cast<FeedbackSlotKind>(slot_kind);
  Handle<FeedbackVector> vector = Handle<FeedbackVector>();
  FeedbackSlot vector_slot = FeedbackSlot::Invalid();
  // This function is only called after looking up in the ScriptContextTable so
  // it is safe to call LoadIC::Load for global loads as well.
  LoadIC ic(isolate, vector, vector_slot, kind);
  ic.UpdateState(receiver, key);
  RETURN_RESULT_OR_FAILURE(isolate, ic.Load(receiver, key));
}
```



```
  0x55d0322a1fd0      00010113       mv         sp, sp               00007f72777eddb0    (387)    int64:140129607802288 uint64:140129607802288
  0x55d0322a1fd4      00090f93       mv         t6, s2               000055d032e3af38    (388)    int64:94352695340856 uint64:94352695340856
  0x55d0322a1fd8      000f80e7       jalr       t6                   000055d0322a1fdc    (389)    int64:94352683179996 uint64:94352683179996
Call to host function Runtime::LoadNoFeedbackIC_Miss at 0x55d0319b4680 args 00000003 , 7f72777eddf0 , 55d032ff8bd0 , 000000fc , bc00000000 , bb00000000 , 000000bc , ffffffffffff
7f4 , 55d0322a1fdc , 000000bc
Returned 00000040  : d92a3de471
  0x55d032e3af38      00000073       ecall
  0x55d0322a1fdc      150b3703       ld         a4, 336(s6)          0000000174c01cb1    (391)    int64:6253714609 uint64:6253714609 <-- [addr: 55d032ff8e20]
```

* V8的simulator 如何调用host function： https://www.bilibili.com/video/BV1hp4y1t7Mx?p=6

Bytecode-1: 间接穿线，Star lookahead，tail dispatch

```
@interpreter/interpreter-generator.cc

void LdaGlobal(int slot_operand_index, int name_operand_index,
               TypeofMode typeof_mode) {
...
   ExitPoint exit_point(this, [=](TNode<Object> result) {
      SetAccumulator(result);
      Dispatch();
   });
...
   accessor_asm.LoadGlobalIC(maybe_feedback_vector, lazy_slot, lazy_context,
                             lazy_name, typeof_mode, &exit_point);
 }
```

Bytecode-2: Star1 （lookaheaded）

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

Bytecode-2: Star1 间接穿线，star lookahead，tail dispatch

@ interpreter/interpreter-assembler.cc

```
void InterpreterAssembler::Dispatch() {
  DCHECK_IMPLIES(Bytecodes::MakesCallAlongCriticalPath(bytecode_), made_call_);
  TNode<IntPtrT> target_offset = Advance();
  TNode<WordT> target_bytecode = LoadBytecode(target_offset);
  DispatchToBytecodeWithOptionalStarLookahead(target_bytecode);
}

void InterpreterAssembler::DispatchToBytecodeWithOptionalStarLookahead(
    TNode<WordT> target_bytecode) {
  if (Bytecodes::IsStarLookahead(bytecode_, operand_scale_)) {
    StarDispatchLookahead(target_bytecode); //处理Star1
  }
  DispatchToBytecode(target_bytecode, BytecodeOffset()); //分发到LdaConstant
}
```

Bytecode-3/4: LdaConstant with Star lookahead

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

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

Bytecode-5: CallUndefinedReceiver1 w/o Star lookahead

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

Bytecode-5: CallUndefinedReceiver1 w/o Star lookahead 返回到InterpreterEntryTrampoline

@ interpreter/interpreter-generator.cc

```
IGNITION_HANDLER(CallUndefinedReceiver1, InterpreterJSCallAssembler) {
  JSCallN(1, ConvertReceiverMode::kNullOrUndefined);
}
```

下面是code assembler的调用链，直接从Interpreter尾调用然后回到InterpreterEntryTrampoline

JSCallN() @interpreter-generator.cc

→CallJSAndDispatch() @interpreter/interpreter-assembler.cc

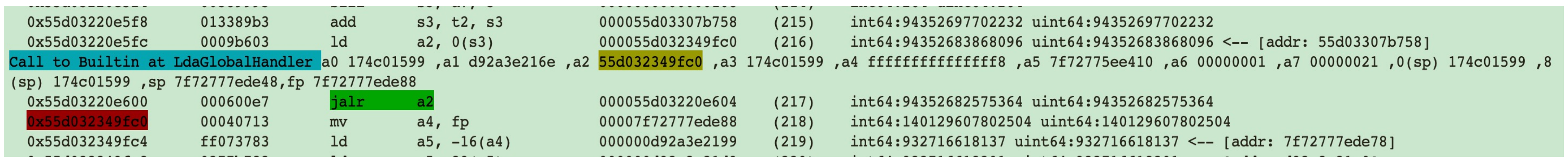
  → TailCallStubThenBytecodeDispatch() @compiler/code-assembler.h

    →TailCallStubThenBytecodeDispatchImpl() @compiler/code-assembler.cc

      →raw_assembler()->TailCallN(call_descriptor, inputs.size(), inputs.data()); @compiler/raw-machine-assembler.cc

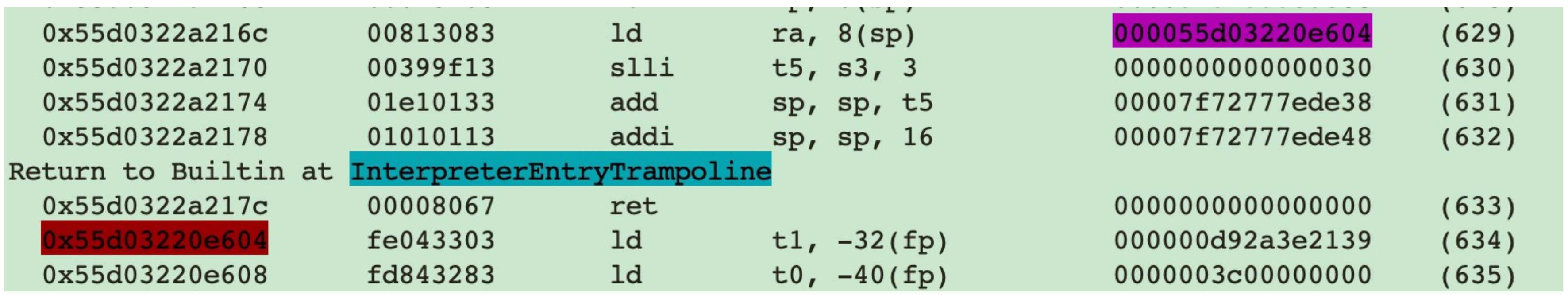
Bytecode-5: CallUndefinedReceiver1 w/o Star lookahead 返回到InterpreterEntryTrampoline

进入InterpreterEntryTrampoline，jalr把返回地址设置到ra中：



```
  0x55d03220e5f8    013389b3    add     s3, t2, s3          000055d03307b758    (215)    int64:94352697702232 uint64:94352697702232
  0x55d03220e5fc    0009b603    ld      a2, 0(s3)           000055d032349fc0    (216)    int64:94352683868096 uint64:94352683868096 <-- [addr: 55d03307b758]
Call to Builtin at LdaGlobalHandler a0 174c01599 ,a1 d92a3e216e ,a2 55d032349fc0 ,a3 174c01599 ,a4 fffffffffffffff8 ,a5 7f72775ee410 ,a6 00000001 ,a7 00000021 ,0(sp) 174c01599 ,8
(sp) 174c01599 ,sp 7f72777ede48,fp 7f72777ede88
  0x55d03220e600    000600e7    jalr        a2                  000055d03220e604    (217)    int64:94352682575364 uint64:94352682575364
  0x55d032349fc0    00040713    mv      a4, fp              00007f72777ede88    (218)    int64:140129607802504 uint64:140129607802504
  0x55d032349fc4    ff073783    ld      a5, -16(a4)         000000d92a3e2199    (219)    int64:932716618137 uint64:932716618137 <-- [addr: 7f72777ede78]
```

返回InterpreterEntryTrampoline，恢复ra，ret返回到InterpreterEntryTrampoline：



```
  0x55d0322a216c    00813083    ld      ra, 8(sp)           000055d03220e604    (629)
  0x55d0322a2170    00399f13    slli    t5, s3, 3           0000000000000030    (630)
  0x55d0322a2174    01e10133    add     sp, sp, t5          00007f72777ede38    (631)
  0x55d0322a2178    01010113    addi    sp, sp, 16          00007f72777ede48    (632)
Return to Builtin at InterpreterEntryTrampoline
  0x55d0322a217c    00008067    ret                         0000000000000000    (633)
  0x55d03220e604    fe043303    ld      t1, -32(fp)         000000d92a3e2139    (634)
  0x55d03220e608    fd843283    ld      t0, -40(fp)         0000003c00000000    (635)
```

Bytecode-6: Star0 w/o Star lookahead

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

Bytecode-7: Return final to InterpreterEntryTrampoline

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

InterpreterEntryTrampoline Return

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

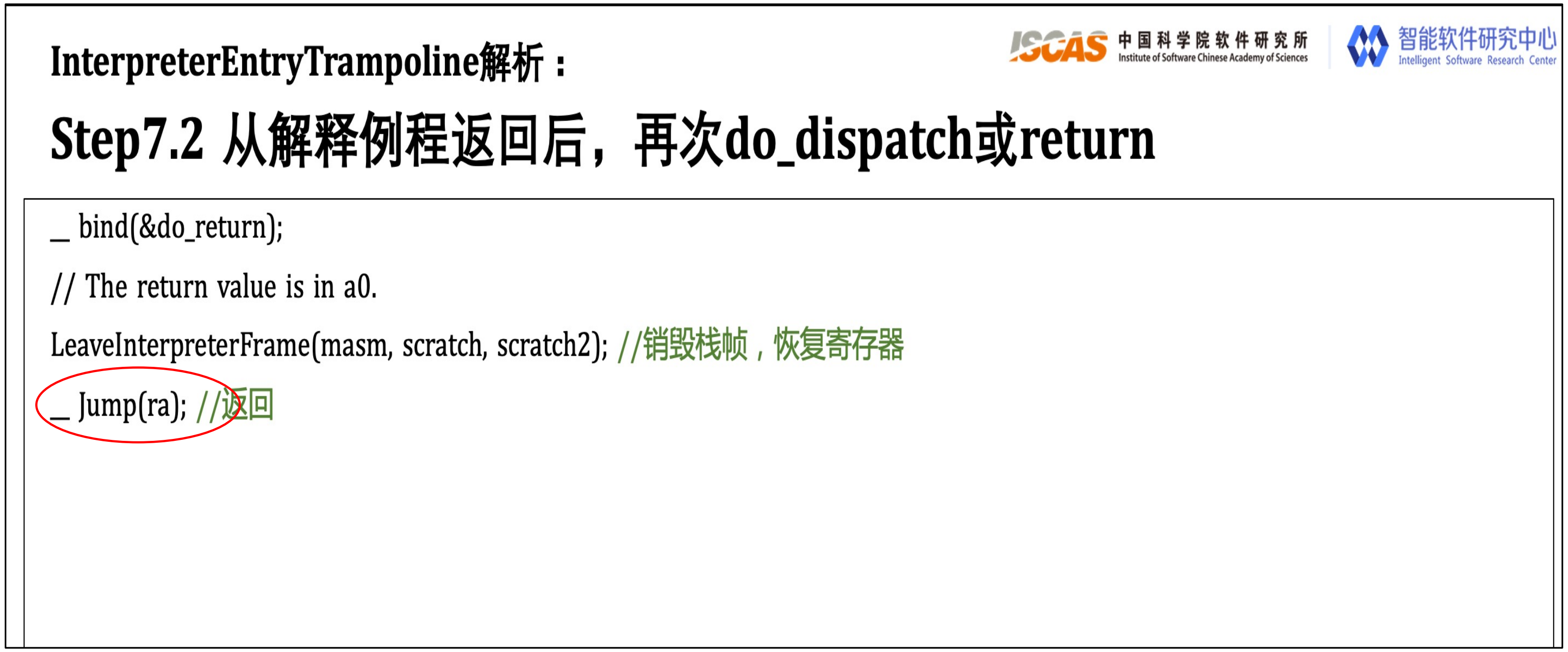
第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

从InterpreterEntryTrampoline返回



InterpreterEntryTrampoline解析：

Step7.2 从解释例程返回后，再次do_dispatch或return

```
_ bind(&do_return);
// The return value is in a0.
LeaveInterpreterFrame(masm, scratch, scratch2); //销毁栈帧，恢复寄存器
_ Jump(ra); //返回
```

* V8中HelloWorld的执行过程-Part3 https://www.bilibili.com/video/BV1hp4y1t7Mx?p=11

JSEntryTrampoline Return

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

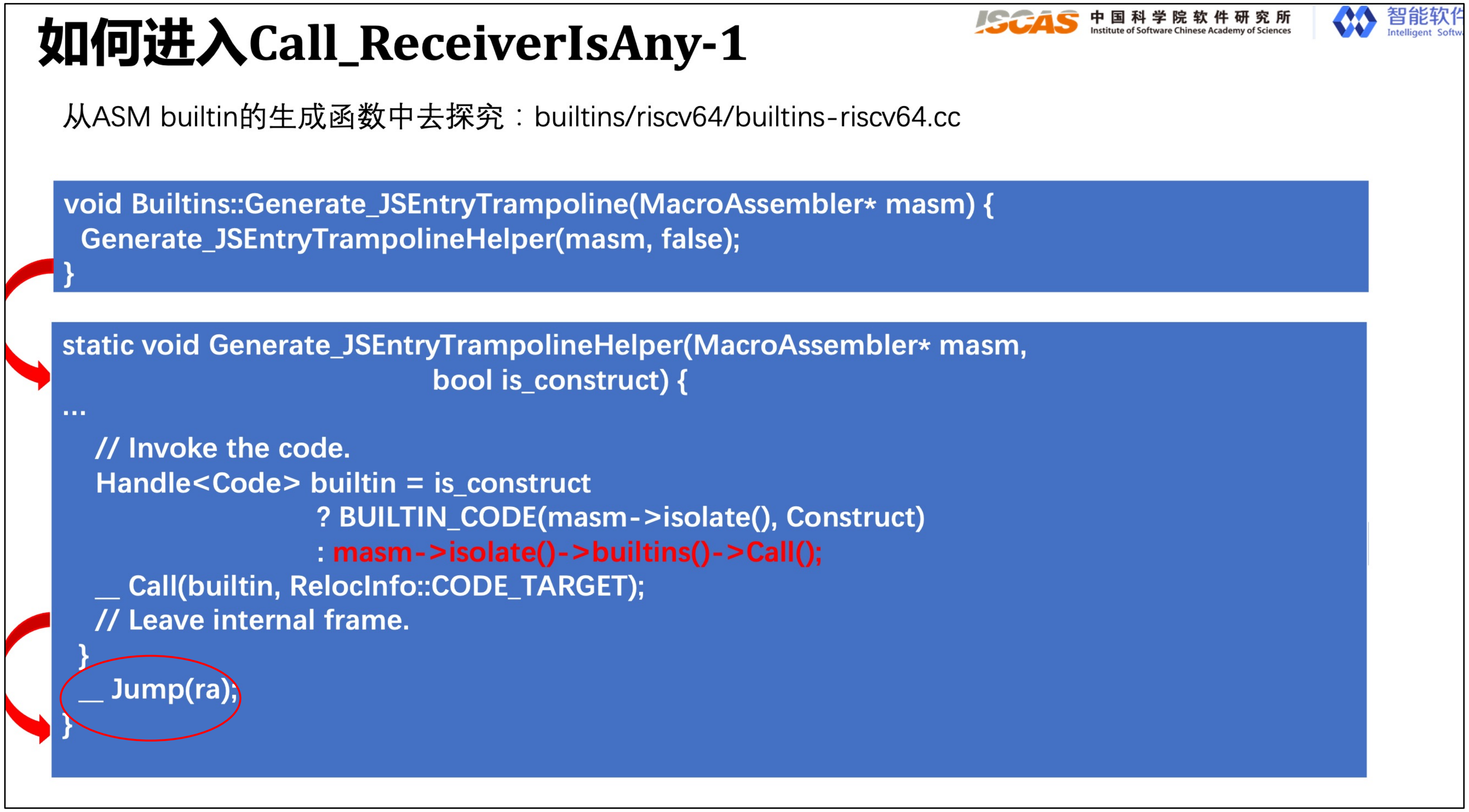
第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

JSEntryTrampoline Return



如何进入Call_ReceiverIsAny-1

从ASM builtin的生成函数中去探究：builtins/riscv64/builtins-riscv64.cc

```
void Builtins::Generate_JSEntryTrampoline(MacroAssembler* masm) {
  Generate_JSEntryTrampolineHelper(masm, false);
}
```

```
static void Generate_JSEntryTrampolineHelper(MacroAssembler* masm,
                                             bool is_construct) {
...
    // Invoke the code.
    Handle<Code> builtin = is_construct
                           ? BUILTIN_CODE(masm->isolate(), Construct)
                           : masm->isolate()->builtins()->Call();
    __ Call(builtin, RelocInfo::CODE_TARGET);
    // Leave internal frame.
  }
  __ Jump(ra);
}
```

* V8中HelloWorld的执行过程-Part2 https://www.bilibili.com/video/BV1hp4y1t7Mx?p=10

JSEntry Return

1	LdaGlobal [0], [0]
2	Star1
3	LdaConstant [1]
4	Star2
5	CallUndefinedReceiver1 r1, r2, [2]
6	Star0
7	Return

第一部分：Prologue

```
CallImpl JSEntry
Call Builtin JSEntryTrampoline
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
```

第二部分：解释器主体

```
Call Builtin InterpreterEntryTrampoline
Call Builtin LdaGlobalHandler
Call Builtin LoadGlobalIC_NoFeedback
Call Builtin LoadIC_NoFeedback
Call Builtin
CEntry_Return1_DontSaveFPRegs_ArgvOnStack_NoBuiltinExit
Call host Runtime::LoadNoFeedbackIC_Miss
Return Builtin LdaGlobalHandler
Call Builtin LdaConstantHandler
Call Builtin CallUndefinedReceiver1Handler
Call Builtin Call_ReceiverIsAny
Call Builtin CallFunction_ReceiverIsAny
Call Builtin HandleApiCall
Call Builtin AdaptorWithBuiltinExitFrame
Call Builtin CEntry_Return1_DontSaveFPRegs_ArgvOnStack_BuiltinExit
Call host Builtin_HandleApiCall
Return Builtin InterpreterEntryTrampoline
Call Builtin ShortStarHandler
Call Builtin ReturnHandler
Return Builtin InterpreterEntryTrampoline
```

第三部分：Epilogue

```
Return Builtin JSEntryTrampoline
Return Builtin JSEntry
```

JSEntry Return

```
@builtins/riscv64/builtins-riscv64.cc

void Generate_JSEntryVariant(MacroAssembler* masm, StackFrame::Type type,
                             Builtin entry_trampoline) {
  Label invoke, handler_entry, exit;
...
// Invoke the function by calling through JS entry trampoline builtin and
  // pop the faked function when we return.
  Handle<Code> trampoline_code =
      masm->isolate()->builtins()->code_handle(entry_trampoline);
  __ Call(trampoline_code, RelocInfo::CODE_TARGET);
  // Unlink this frame from the handler chain.
  __ PopStackHandler();
...
  // Return.
  __ Jump(ra);
}
```

* V8中HelloWorld的执行过程-Part2 https://www.bilibili.com/video/BV1hp4y1t7Mx?p=10

总结（V8 helloworld执行过程完结）

- 在part1中，讲述了：
  - hello.js：print(“HelloWorld!”)的字节码和含义
  - 如何从--trace-sim的log文件中， 梳理hello.js的解释执行过程
- 在part2中，讲述了：
  - d8上hello.js的整体执行流程
  - 如何进入第一部分Prologue部分开始执行
    - Builtin by Builtin
- 在part3中，讲述了：
  - Ignition解释循环主体 InterpreterEntryTrampoline的整体控制流程
- part4中，讲述了：
  - Bytecode by Bytecode的解释执行流程和返回过程
- 作业：
  - 从JSEntry返回后，是如何又回到V8的host部分执行的呢？（答案在slides致谢页后）

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

谢 谢

欢迎交流合作

2020/11/12

如何从JSEntry返回到host的执行

```
void Simulator::CallInternal(Address entry) { //v8i: final kick before RISV64 binary simulation
  // Adjust JS-based stack limit to C-based stack limit.
  isolate_->stack_guard()->AdjustStackLimitForSimulator();


  // Prepare to execute the code at entry.
  set_register(pc, static_cast<int64_t>(entry));
  // Put down marker for end of simulation. The simulator will stop simulation
  // when the PC reaches this value. By saving the "end simulation" value into
  // the LR the simulation stops when returning to this call point.
  set_register(ra, end_sim_pc); //将ra设置成end_sim_pc
...
}
```

如何从JSEntry返回到host的执行

```
void Simulator::Execute() {
  // Get the PC to simulate. Cannot use the accessor here as we need the
  // raw PC value and not the one used as input to arithmetic instructions.
  int64_t program_counter = get_pc(); //v8i: first PC is to JSEntry Builtin
  while (program_counter != end_sim_pc) { //如果遇到end_sim_pc就停止模拟，返回到CallInternal
    Instruction* instr = reinterpret_cast<Instruction*>(program_counter);
    icount_++;
    if (icount_ == static_cast<int64_t>(::v8::internal::FLAG_stop_sim_at)) {
      RiscvDebugger dbg(this);
      dbg.Debug();
    } else {
      InstructionDecode(instr);
    }
    CheckBreakpoints();
    program_counter = get_pc();
  }
}
```

如何进入JSEntry-3

```
@  v8::internal::(anonymous namespace)::Invoke (isolate=0x55b8725c59a0, params=...)
```

```
@ v8::internal::GeneratedCode<unsigned long, unsigned long, unsigned long, unsigned long,
```

```
@ v8::internal::Simulator::Call<unsigned long, unsigned long, unsigned long, unsigned long,
unsigned long, long, unsigned long**> (this=0x55f6c89be450, entry=140326543809824,
    args=0x0, args=0x0, args=0x0, args=0x0, args=0x0, args=0x0)
at ../../src/execution/riscv64/simulator-riscv64.h:369

template <typename Return, typename... Args>
  Return Call(Address entry, Args... args) {
    return VariadicCall<Return>(this, &Simulator::CallImpl, entry, args...);
  }
```

```
@ v8::internal::SimulatorBase::VariadicCall at ../../src/execution/simulator-base.h:46
```

```
@ v8::internal::Simulator::CallImpl (this=0x55f6c89be450, entry=140326543809824,
argument_count=6, arguments=0x7ffc32118c30) at ../../src/execution/riscv64/simulator-
riscv64.cc:3575
```

```
@ src/execution/riscv64/simulator-riscv64.cc
CallInternal(entry);   -> Execute() // Start the simulation.
```