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

关于ignition中bytecode的总结

软件所智能软件中心PLCT实验室 张江涛 实习生

2020/02/13

目 录

01 Bytecode介绍

- 背景

　　V8的full-codegen编译器生成的机器码是很冗余的，它使V8堆内存的占用在普通网页中明显增加。

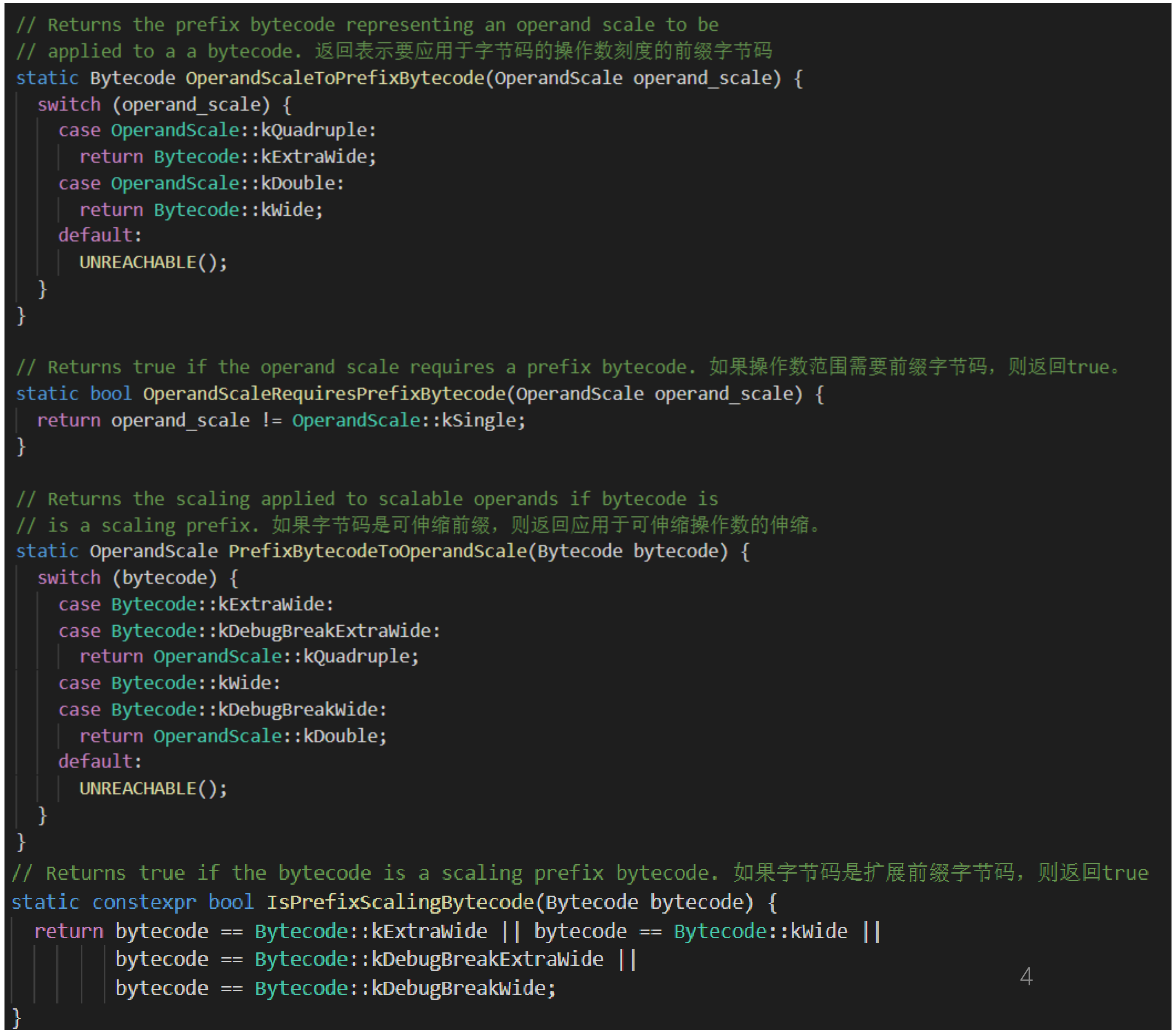
- V8引入Bytecode的意义
  - 减少内存的使用
  - 减少解析开销
  - 减小编译的复杂性

02 Bytecode相关代码

Wide operands

Ignition使用前置特定的字节码来支持wide operands。
Ignition支持固定宽度操作数和可缩放宽度操作数，可缩放的操作数是根据前置字节码按比例缩放的。

- Wide前置字节码变成16位宽（2倍）
- ExtraWide前置字节码变成32位（4倍）

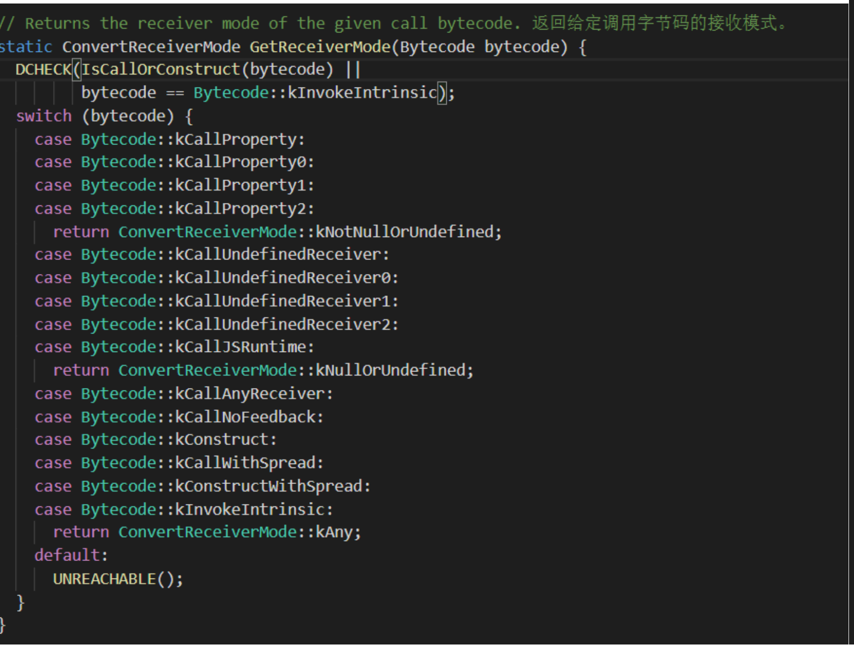


```
// Returns the prefix bytecode representing an operand scale to be
// applied to a a bytecode. 返回表示要应用于字节码的操作数刻度的前缀字节码
static Bytecode OperandScaleToPrefixBytecode(OperandScale operand_scale) {
  switch (operand_scale) {
    case OperandScale::kQuadruple:
      return Bytecode::kExtraWide;
    case OperandScale::kDouble:
      return Bytecode::kWide;
    default:
      UNREACHABLE();
  }
}

// Returns true if the operand scale requires a prefix bytecode. 如果操作数范围需要前缀字节码，则返回true。
static bool OperandScaleRequiresPrefixBytecode(OperandScale operand_scale) {
  return operand_scale != OperandScale::kSingle;
}

// Returns the scaling applied to scalable operands if bytecode is
// is a scaling prefix. 如果字节码是可伸缩前缀，则返回应用于可伸缩操作数的伸缩。
static OperandScale PrefixBytecodeToOperandScale(Bytecode bytecode) {
  switch (bytecode) {
    case Bytecode::kExtraWide:
    case Bytecode::kDebugBreakExtraWide:
      return OperandScale::kQuadruple;
    case Bytecode::kWide:
    case Bytecode::kDebugBreakWide:
      return OperandScale::kDouble;
    default:
      UNREACHABLE();
  }
}
// Returns true if the bytecode is a scaling prefix bytecode. 如果字节码是扩展前缀字节码，则返回true
static constexpr bool IsPrefixScalingBytecode(Bytecode bytecode) {
  return bytecode == Bytecode::kExtraWide || bytecode == Bytecode::kWide ||
         bytecode == Bytecode::kDebugBreakExtraWide ||
         bytecode == Bytecode::kDebugBreakWide;
}
```

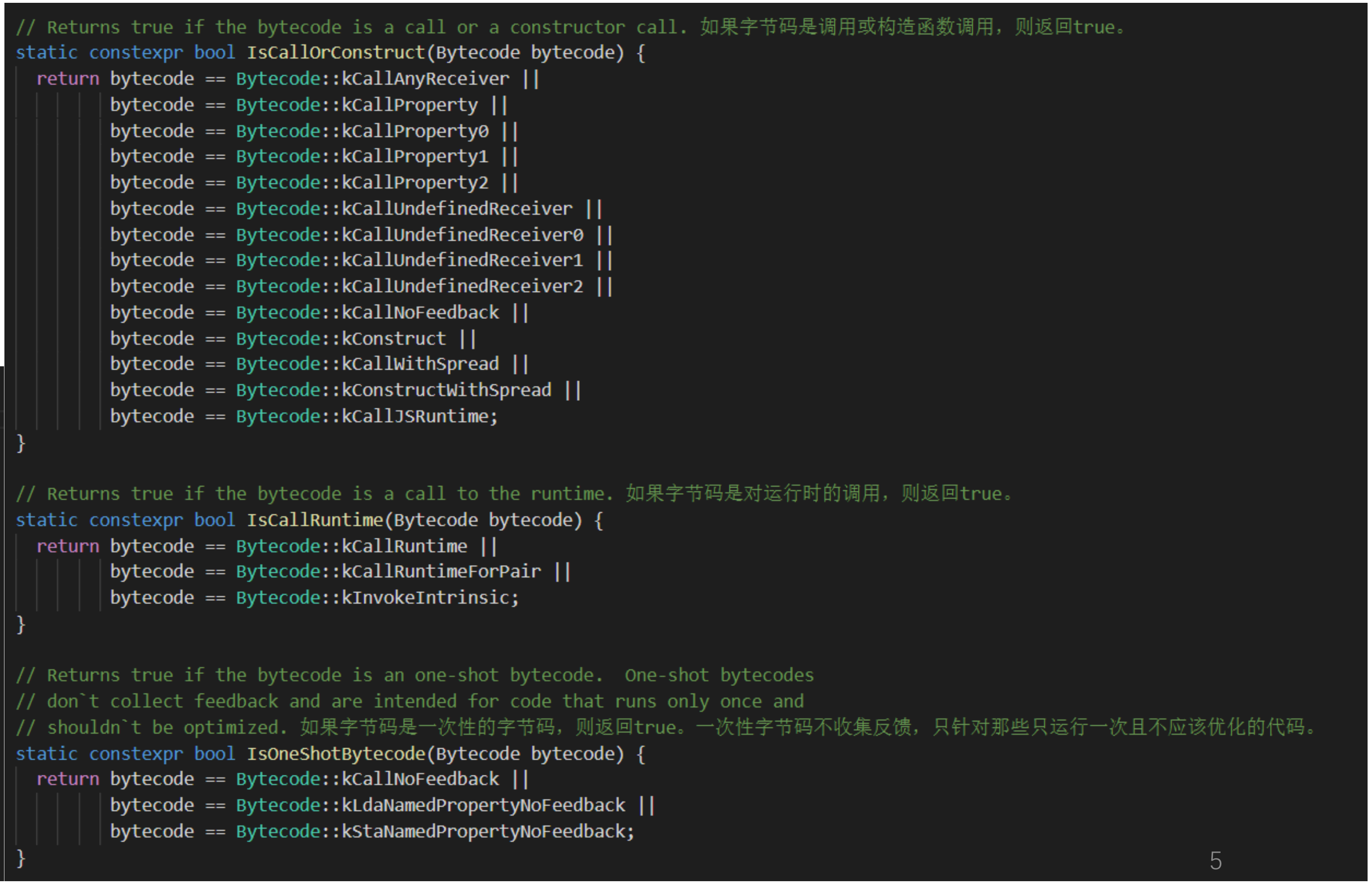
02 Bytecode相关代码

JS call

调用其它的JS函数由Call字节码处理。Call字节码含有三个寄存器操作数。
第一个：被调用者的寄存器
第二个：参数的起始位置寄存器
第三个：传递的参数数量



```
// Returns the receiver mode of the given call bytecode. 返回给定调用字节码的接收模式。
static ConvertReceiverMode GetReceiverMode(Bytecode bytecode) {
  DCHECK(IsCallOrConstruct(bytecode) ||
         bytecode == Bytecode::kInvokeIntrinsic);
  switch (bytecode) {
    case Bytecode::kCallProperty:
    case Bytecode::kCallProperty0:
    case Bytecode::kCallProperty1:
    case Bytecode::kCallProperty2:
      return ConvertReceiverMode::kNotNullOrUndefined;
    case Bytecode::kCallUndefinedReceiver:
    case Bytecode::kCallUndefinedReceiver0:
    case Bytecode::kCallUndefinedReceiver1:
    case Bytecode::kCallUndefinedReceiver2:
    case Bytecode::kCallJSRuntime:
      return ConvertReceiverMode::kNullOrUndefined;
    case Bytecode::kCallAnyReceiver:
    case Bytecode::kCallNoFeedback:
    case Bytecode::kConstruct:
    case Bytecode::kCallWithSpread:
    case Bytecode::kConstructWithSpread:
    case Bytecode::kInvokeIntrinsic:
      return ConvertReceiverMode::kAny;
    default:
      UNREACHABLE();
  }
}
```

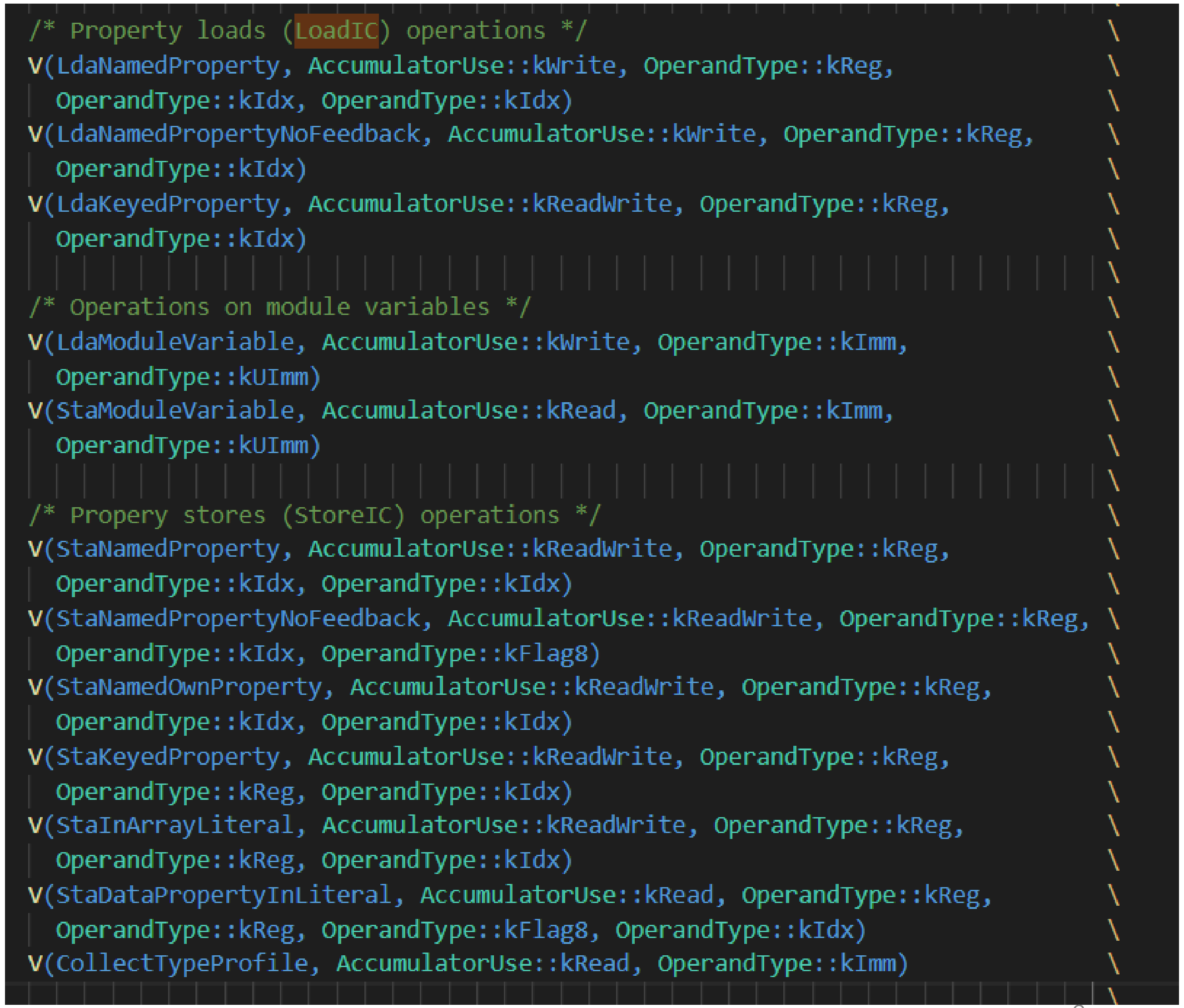


```
// Returns true if the bytecode is a call or a constructor call. 如果字节码是调用或构造函数调用，则返回true。
static constexpr bool IsCallOrConstruct(Bytecode bytecode) {
  return bytecode == Bytecode::kCallAnyReceiver ||
         bytecode == Bytecode::kCallProperty ||
         bytecode == Bytecode::kCallProperty0 ||
         bytecode == Bytecode::kCallProperty1 ||
         bytecode == Bytecode::kCallProperty2 ||
         bytecode == Bytecode::kCallUndefinedReceiver ||
         bytecode == Bytecode::kCallUndefinedReceiver0 ||
         bytecode == Bytecode::kCallUndefinedReceiver1 ||
         bytecode == Bytecode::kCallUndefinedReceiver2 ||
         bytecode == Bytecode::kCallNoFeedback ||
         bytecode == Bytecode::kConstruct ||
         bytecode == Bytecode::kCallWithSpread ||
         bytecode == Bytecode::kConstructWithSpread ||
         bytecode == Bytecode::kCallJSRuntime;
}

// Returns true if the bytecode is a call to the runtime. 如果字节码是对运行时的调用，则返回true。
static constexpr bool IsCallRuntime(Bytecode bytecode) {
  return bytecode == Bytecode::kCallRuntime ||
         bytecode == Bytecode::kCallRuntimeForPair ||
         bytecode == Bytecode::kInvokeIntrinsic;
}

// Returns true if the bytecode is an one-shot bytecode.  One-shot bytecodes
// don`t collect feedback and are intended for code that runs only once and
// shouldn`t be optimized. 如果字节码是一次性的字节码，则返回true。一次性字节码不收集反馈，只针对那些只运行一次且不应该优化的代码。
static constexpr bool IsOneShotBytecode(Bytecode bytecode) {
  return bytecode == Bytecode::kCallNoFeedback ||
         bytecode == Bytecode::kLdaNamedPropertyNoFeedback ||
         bytecode == Bytecode::kStaNamedPropertyNoFeedback;
}
```

02 Bytecode相关代码

Property loads / stores

字节码通过内联缓存（IC）在JS对象上加载和存储属性。字节码处理程序调用相同的LoadIC和StoreIC代码stub（在TypeFeedbackVector上操作）。

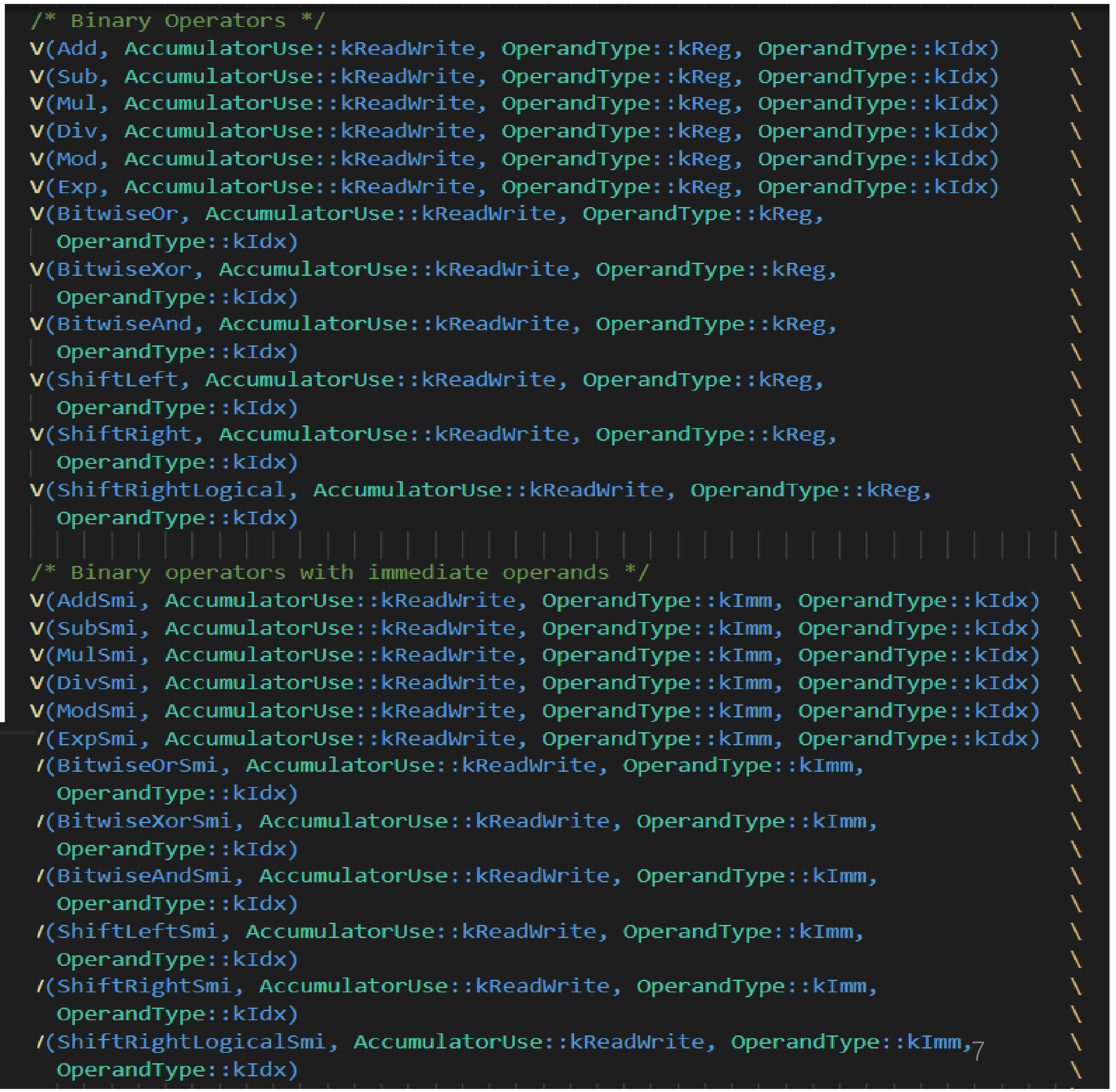


```
/* Property loads (LoadIC) operations */                                    \
V(LdaNamedProperty, AccumulatorUse::kWrite, OperandType::kReg,              \
  OperandType::kIdx, OperandType::kIdx)                                     \
V(LdaNamedPropertyNoFeedback, AccumulatorUse::kWrite, OperandType::kReg,    \
  OperandType::kIdx)                                                        \
V(LdaKeyedProperty, AccumulatorUse::kReadWrite, OperandType::kReg,          \
  OperandType::kIdx)                                                        \
                                                                            \
/* Operations on module variables */                                        \
V(LdaModuleVariable, AccumulatorUse::kWrite, OperandType::kImm,             \
  OperandType::kUImm)                                                       \
V(StaModuleVariable, AccumulatorUse::kRead, OperandType::kImm,              \
  OperandType::kUImm)                                                       \
                                                                            \
/* Propery stores (StoreIC) operations */                                   \
V(StaNamedProperty, AccumulatorUse::kReadWrite, OperandType::kReg,          \
  OperandType::kIdx, OperandType::kIdx)                                     \
V(StaNamedPropertyNoFeedback, AccumulatorUse::kReadWrite, OperandType::kReg, \
  OperandType::kIdx, OperandType::kFlag8)                                   \
V(StaNamedOwnProperty, AccumulatorUse::kReadWrite, OperandType::kReg,       \
  OperandType::kIdx, OperandType::kIdx)                                     \
V(StaKeyedProperty, AccumulatorUse::kReadWrite, OperandType::kReg,          \
  OperandType::kReg, OperandType::kIdx)                                     \
V(StaInArrayLiteral, AccumulatorUse::kReadWrite, OperandType::kReg,         \
  OperandType::kReg, OperandType::kIdx)                                     \
V(StaDataPropertyInLiteral, AccumulatorUse::kRead, OperandType::kReg,       \
  OperandType::kReg, OperandType::kFlag8, OperandType::kIdx)                \
V(CollectTypeProfile, AccumulatorUse::kRead, OperandType::kImm)             \
```

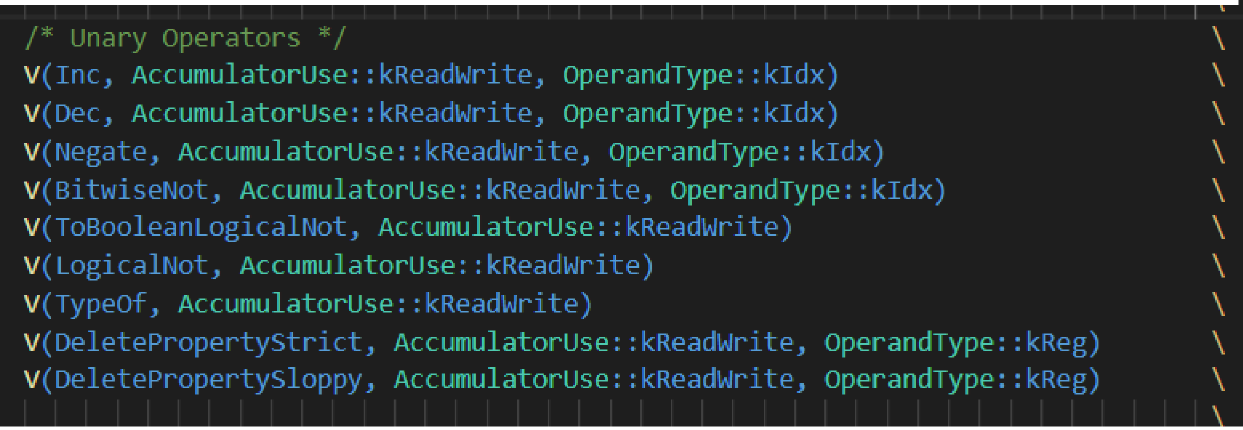
02 Bytecode相关代码

Binary 和 Unary ops

目前没有为二元和一元操作收集任何类型反馈。



```
/* Binary Operators */                                                      \
V(Add, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(Sub, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(Mul, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(Div, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(Mod, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(Exp, AccumulatorUse::kReadWrite, OperandType::kReg, OperandType::kIdx)     \
V(BitwiseOr, AccumulatorUse::kReadWrite, OperandType::kReg,                  \
  OperandType::kIdx)                                                         \
V(BitwiseXor, AccumulatorUse::kReadWrite, OperandType::kReg,                 \
  OperandType::kIdx)                                                         \
V(BitwiseAnd, AccumulatorUse::kReadWrite, OperandType::kReg,                 \
  OperandType::kIdx)                                                         \
V(ShiftLeft, AccumulatorUse::kReadWrite, OperandType::kReg,                  \
  OperandType::kIdx)                                                         \
V(ShiftRight, AccumulatorUse::kReadWrite, OperandType::kReg,                 \
  OperandType::kIdx)                                                         \
V(ShiftRightLogical, AccumulatorUse::kReadWrite, OperandType::kReg,          \
  OperandType::kIdx)                                                         \
                                                                             \
/* Binary operators with immediate operands */                               \
V(AddSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(SubSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(MulSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(DivSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(ModSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(ExpSmi, AccumulatorUse::kReadWrite, OperandType::kImm, OperandType::kIdx)  \
V(BitwiseOrSmi, AccumulatorUse::kReadWrite, OperandType::kImm,               \
  OperandType::kIdx)                                                         \
V(BitwiseXorSmi, AccumulatorUse::kReadWrite, OperandType::kImm,              \
  OperandType::kIdx)                                                         \
V(BitwiseAndSmi, AccumulatorUse::kReadWrite, OperandType::kImm,              \
  OperandType::kIdx)                                                         \
V(ShiftLeftSmi, AccumulatorUse::kReadWrite, OperandType::kImm,               \
  OperandType::kIdx)                                                         \
V(ShiftRightSmi, AccumulatorUse::kReadWrite, OperandType::kImm,              \
  OperandType::kIdx)                                                         \
V(ShiftRightLogicalSmi, AccumulatorUse::kReadWrite, OperandType::kImm,       \
  OperandType::kIdx)                                                         \
```

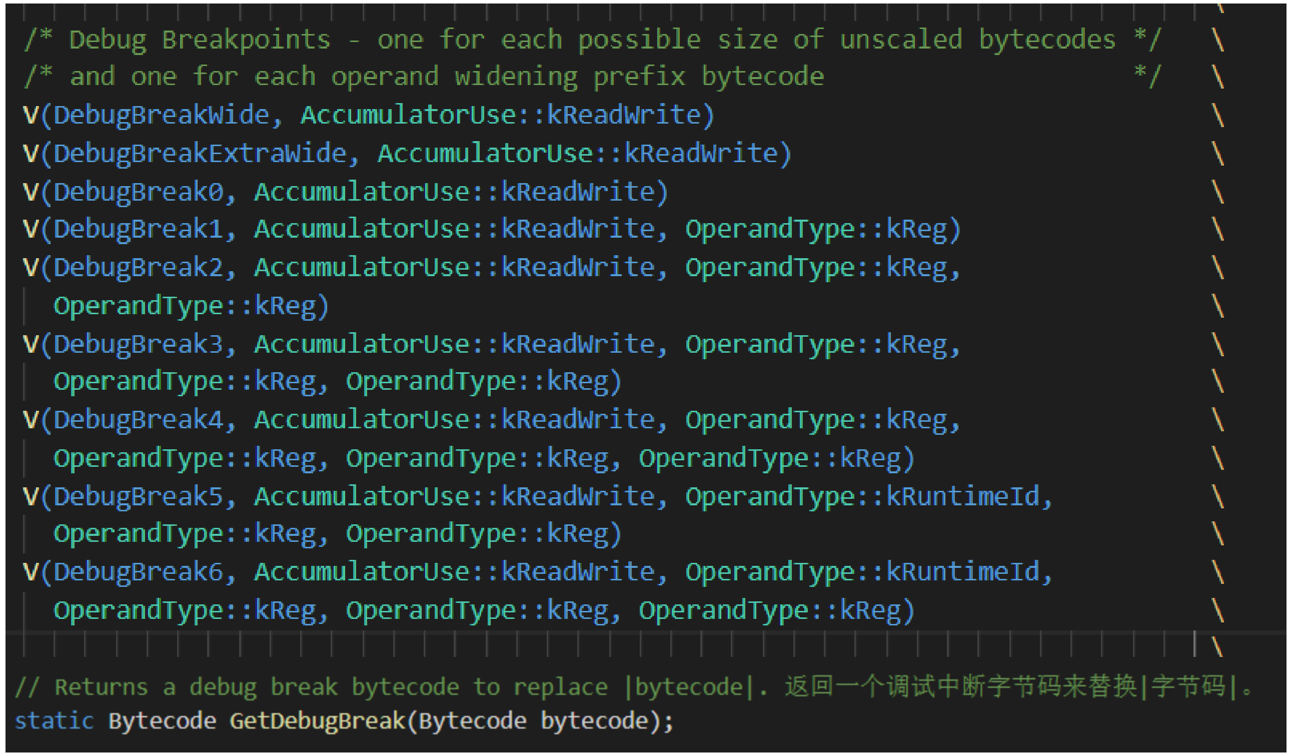


```
/* Unary Operators */                                                         \
V(Inc, AccumulatorUse::kReadWrite, OperandType::kIdx)                         \
V(Dec, AccumulatorUse::kReadWrite, OperandType::kIdx)                         \
V(Negate, AccumulatorUse::kReadWrite, OperandType::kIdx)                      \
V(BitwiseNot, AccumulatorUse::kReadWrite, OperandType::kIdx)                  \
V(ToBooleanLogicalNot, AccumulatorUse::kReadWrite)                            \
V(LogicalNot, AccumulatorUse::kReadWrite)                                     \
V(TypeOf, AccumulatorUse::kReadWrite)                                         \
V(DeletePropertyStrict, AccumulatorUse::kReadWrite, OperandType::kReg)        \
V(DeletePropertySloppy, AccumulatorUse::kReadWrite, OperandType::kReg)        \
                                                                              \
```

02 Bytecode相关代码

Debugging support

为了支持解释器执行时调试器断点代码，调试器复制函数中BytecodeArray对象，然后讲断点目标位置的任意字节码替换为特定的DebugBreak字节码。对于每种大小的字节码，有不同的DebugBreak字节码变种来保证BytecodeArray仍然可以正确迭代。



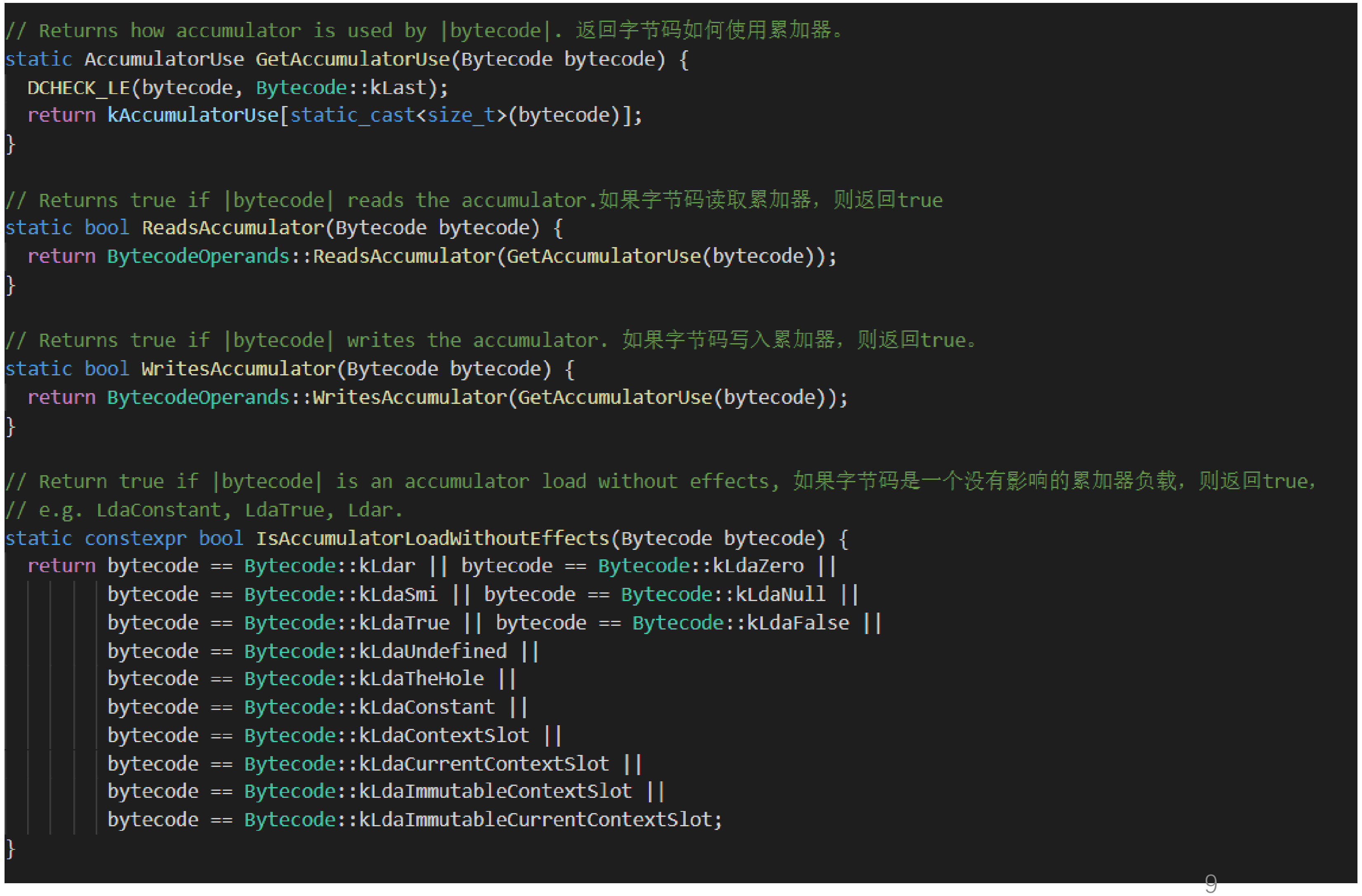
```
/* Debug Breakpoints - one for each possible size of unscaled bytecodes */   \
/* and one for each operand widening prefix bytecode                    */   \
V(DebugBreakWide, AccumulatorUse::kReadWrite)                               \
V(DebugBreakExtraWide, AccumulatorUse::kReadWrite)                          \
V(DebugBreak0, AccumulatorUse::kReadWrite)                                  \
V(DebugBreak1, AccumulatorUse::kReadWrite, OperandType::kReg)               \
V(DebugBreak2, AccumulatorUse::kReadWrite, OperandType::kReg,               \
  OperandType::kReg)                                                        \
V(DebugBreak3, AccumulatorUse::kReadWrite, OperandType::kReg,               \
  OperandType::kReg, OperandType::kReg)                                     \
V(DebugBreak4, AccumulatorUse::kReadWrite, OperandType::kReg,               \
  OperandType::kReg, OperandType::kReg, OperandType::kReg)                  \
V(DebugBreak5, AccumulatorUse::kReadWrite, OperandType::kRuntimeId,         \
  OperandType::kReg, OperandType::kReg)                                     \
V(DebugBreak6, AccumulatorUse::kReadWrite, OperandType::kRuntimeId,         \
  OperandType::kReg, OperandType::kReg, OperandType::kReg)                  \
                                                                            \
// Returns a debug break bytecode to replace |bytecode|. 返回一个调试中断字节码来替换|字节码|。
static Bytecode GetDebugBreak(Bytecode bytecode);
```

02 Bytecode相关代码

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

Accumulator

关于累加器的使用主要分为三种:
1、累加器的读取
2、累加器的写入
3、没有影响的累加器负载

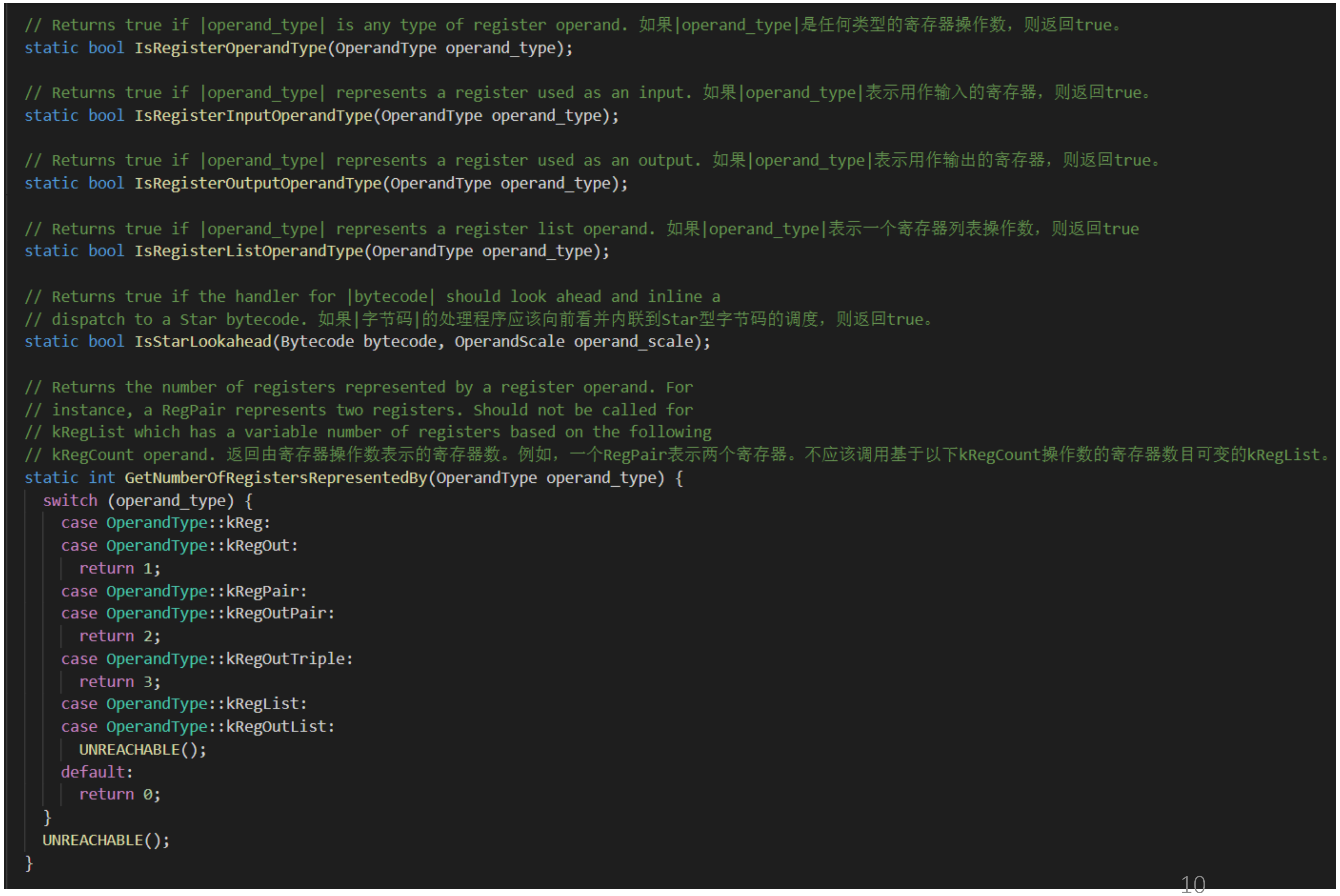


```
// Returns how accumulator is used by |bytecode|. 返回字节码如何使用累加器。
static AccumulatorUse GetAccumulatorUse(Bytecode bytecode) {
  DCHECK_LE(bytecode, Bytecode::kLast);
  return kAccumulatorUse[static_cast<size_t>(bytecode)];
}

// Returns true if |bytecode| reads the accumulator.如果字节码读取累加器，则返回true
static bool ReadsAccumulator(Bytecode bytecode) {
  return BytecodeOperands::ReadsAccumulator(GetAccumulatorUse(bytecode));
}

// Returns true if |bytecode| writes the accumulator. 如果字节码写入累加器，则返回true。
static bool WritesAccumulator(Bytecode bytecode) {
  return BytecodeOperands::WritesAccumulator(GetAccumulatorUse(bytecode));
}

// Return true if |bytecode| is an accumulator load without effects, 如果字节码是一个没有影响的累加器负载，则返回true，
// e.g. LdaConstant, LdaTrue, Ldar.
static constexpr bool IsAccumulatorLoadWithoutEffects(Bytecode bytecode) {
  return bytecode == Bytecode::kLdar || bytecode == Bytecode::kLdaZero ||
         bytecode == Bytecode::kLdaSmi || bytecode == Bytecode::kLdaNull ||
         bytecode == Bytecode::kLdaTrue || bytecode == Bytecode::kLdaFalse ||
         bytecode == Bytecode::kLdaUndefined ||
         bytecode == Bytecode::kLdaTheHole ||
         bytecode == Bytecode::kLdaConstant ||
         bytecode == Bytecode::kLdaContextSlot ||
         bytecode == Bytecode::kLdaCurrentContextSlot ||
         bytecode == Bytecode::kLdaImmutableContextSlot ||
         bytecode == Bytecode::kLdaImmutableCurrentContextSlot;
}
```

02 Bytecode相关代码

Register

关于寄存器的使用:
1、寄存器的读取
2、寄存器的写入
3、寄存器列表操作数
4、返回寄存器操作数表示的寄存器数。
5、没有影响的寄存器器负载



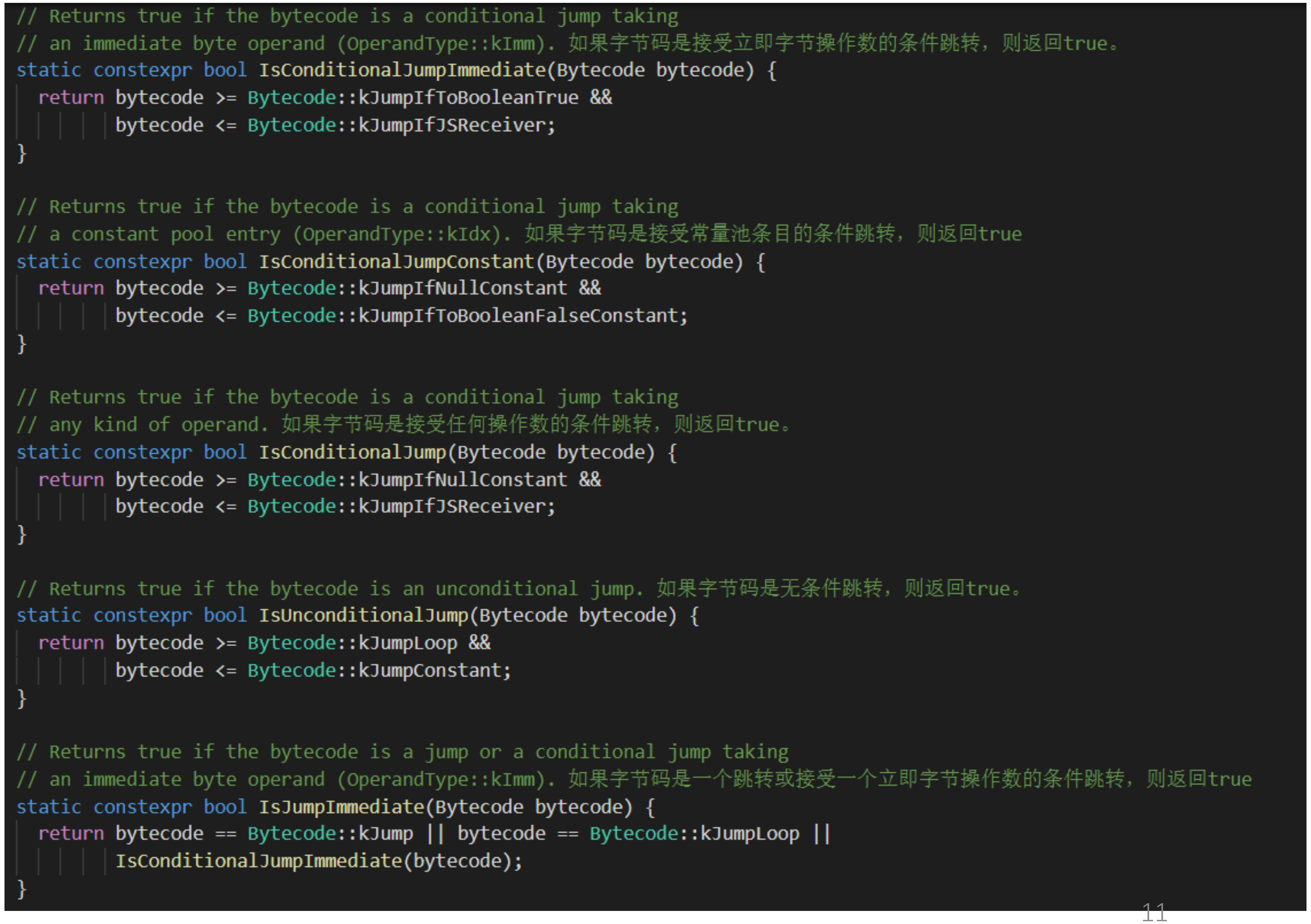
```
// Returns true if |operand_type| is any type of register operand. 如果|operand_type|是任何类型的寄存器操作数，则返回true。
static bool IsRegisterOperandType(OperandType operand_type);

// Returns true if |operand_type| represents a register used as an input. 如果|operand_type|表示用作输入的寄存器，则返回true。
static bool IsRegisterInputOperandType(OperandType operand_type);

// Returns true if |operand_type| represents a register used as an output. 如果|operand_type|表示用作输出的寄存器，则返回true。
static bool IsRegisterOutputOperandType(OperandType operand_type);

// Returns true if |operand_type| represents a register list operand. 如果|operand_type|表示一个寄存器列表操作数，则返回true
static bool IsRegisterListOperandType(OperandType operand_type);

// Returns true if the handler for |bytecode| should look ahead and inline a
// dispatch to a Star bytecode. 如果|字节码|的处理程序应该向前看并内联到Star型字节码的调度，则返回true。
static bool IsStarLookahead(Bytecode bytecode, OperandScale operand_scale);

// Returns the number of registers represented by a register operand. For
// instance, a RegPair represents two registers. Should not be called for
// kRegList which has a variable number of registers based on the following
// kRegCount operand. 返回由寄存器操作数表示的寄存器数。例如，一个RegPair表示两个寄存器。不应该调用基于以下kRegCount操作数的寄存器数目可变的kRegList。
static int GetNumberOfRegistersRepresentedBy(OperandType operand_type) {
  switch (operand_type) {
    case OperandType::kReg:
    case OperandType::kRegOut:
      return 1;
    case OperandType::kRegPair:
    case OperandType::kRegOutPair:
      return 2;
    case OperandType::kRegOutTriple:
      return 3;
    case OperandType::kRegList:
    case OperandType::kRegOutList:
      UNREACHABLE();
    default:
      return 0;
  }
  UNREACHABLE();
}
```

02 Bytecode相关代码

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

Jump

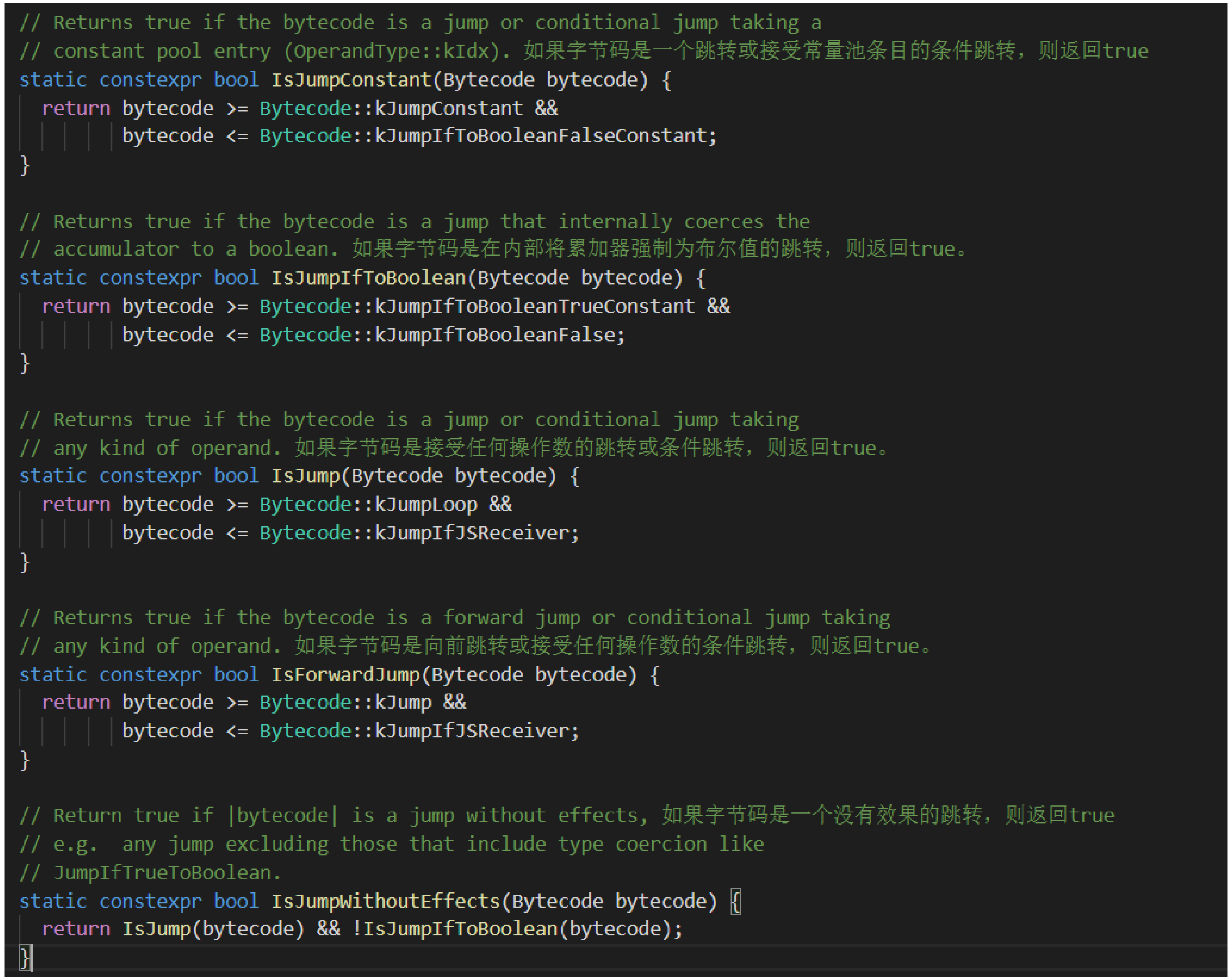
关于控制流字节码，当满足一定的条件，就会跳转到满足条件的程序块中去执行相应的操作。用于控制整个程序的一个动态。



```
// Returns true if the bytecode is a conditional jump taking
// an immediate byte operand (OperandType::kImm). 如果字节码是接受立即字节操作数的条件跳转，则返回true。
static constexpr bool IsConditionalJumpImmediate(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpIfToBooleanTrue &&
         bytecode <= Bytecode::kJumpIfJSReceiver;
}

// Returns true if the bytecode is a conditional jump taking
// a constant pool entry (OperandType::kIdx). 如果字节码是接受常量池条目的条件跳转，则返回true
static constexpr bool IsConditionalJumpConstant(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpIfNullConstant &&
         bytecode <= Bytecode::kJumpIfToBooleanFalseConstant;
}

// Returns true if the bytecode is a conditional jump taking
// any kind of operand. 如果字节码是接受任何操作数的条件跳转，则返回true。
static constexpr bool IsConditionalJump(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpIfNullConstant &&
         bytecode <= Bytecode::kJumpIfJSReceiver;
}

// Returns true if the bytecode is an unconditional jump. 如果字节码是无条件跳转，则返回true。
static constexpr bool IsUnconditionalJump(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpLoop &&
         bytecode <= Bytecode::kJumpConstant;
}

// Returns true if the bytecode is a jump or a conditional jump taking
// an immediate byte operand (OperandType::kImm). 如果字节码是一个跳转或接受一个立即字节操作数的条件跳转，则返回true
static constexpr bool IsJumpImmediate(Bytecode bytecode) {
  return bytecode == Bytecode::kJump || bytecode == Bytecode::kJumpLoop ||
         IsConditionalJumpImmediate(bytecode);
}
```

02 Bytecode相关代码

Jump

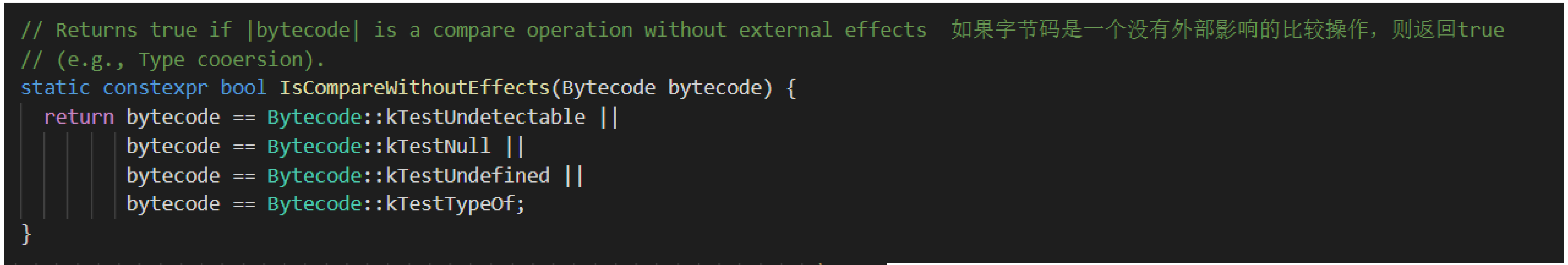


```
// Returns true if the bytecode is a jump or conditional jump taking a
// constant pool entry (OperandType::kIdx). 如果字节码是一个跳转或接受常量池条目的条件跳转，则返回true
static constexpr bool IsJumpConstant(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpConstant &&
         bytecode <= Bytecode::kJumpIfToBooleanFalseConstant;
}

// Returns true if the bytecode is a jump that internally coerces the
// accumulator to a boolean. 如果字节码是在内部将累加器强制为布尔值的跳转，则返回true。
static constexpr bool IsJumpIfToBoolean(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpIfToBooleanTrueConstant &&
         bytecode <= Bytecode::kJumpIfToBooleanFalse;
}

// Returns true if the bytecode is a jump or conditional jump taking
// any kind of operand. 如果字节码是接受任何操作数的跳转或条件跳转，则返回true。
static constexpr bool IsJump(Bytecode bytecode) {
  return bytecode >= Bytecode::kJumpLoop &&
         bytecode <= Bytecode::kJumpIfJSReceiver;
}

// Returns true if the bytecode is a forward jump or conditional jump taking
// any kind of operand. 如果字节码是向前跳转或接受任何操作数的条件跳转，则返回true。
static constexpr bool IsForwardJump(Bytecode bytecode) {
  return bytecode >= Bytecode::kJump &&
         bytecode <= Bytecode::kJumpIfJSReceiver;
}

// Return true if |bytecode| is a jump without effects, 如果字节码是一个没有效果的跳转，则返回true
// e.g.  any jump excluding those that include type coercion like
// JumpIfTrueToBoolean.
static constexpr bool IsJumpWithoutEffects(Bytecode bytecode) {
  return IsJump(bytecode) && !IsJumpIfToBoolean(bytecode);
}
```

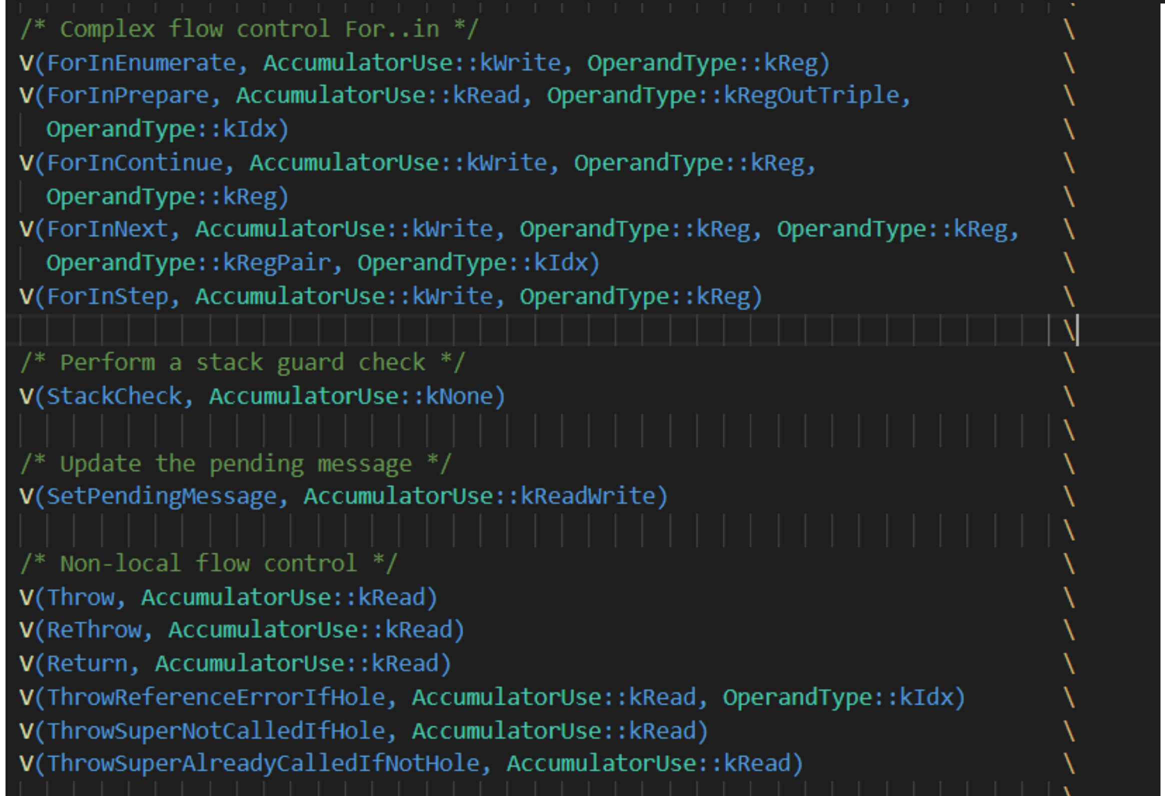
02 Bytecode相关代码

other

- 栈检查保护
- For…in支持
- 非局部控制流
- 非法字节
- Test操作
- 类型转换
- 参数支持
- New操作
- 内部函数
- 闭包申请
- 参数申请
- literals



```
// Returns true if |bytecode| is a compare operation without external effects  如果字节码是一个没有外部影响的比较操作，则返回true
// (e.g., Type cooersion).
static constexpr bool IsCompareWithoutEffects(Bytecode bytecode) {
  return bytecode == Bytecode::kTestUndetectable ||
         bytecode == Bytecode::kTestNull ||
         bytecode == Bytecode::kTestUndefined ||
         bytecode == Bytecode::kTestTypeOf;
}
```



```
/* Complex flow control For..in */                                          \
V(ForInEnumerate, AccumulatorUse::kWrite, OperandType::kReg)                 \
V(ForInPrepare, AccumulatorUse::kRead, OperandType::kRegOutTriple,           \
  OperandType::kIdx)                                                         \
V(ForInContinue, AccumulatorUse::kWrite, OperandType::kReg,                  \
  OperandType::kReg)                                                         \
V(ForInNext, AccumulatorUse::kWrite, OperandType::kReg, OperandType::kReg,   \
  OperandType::kRegPair, OperandType::kIdx)                                  \
V(ForInStep, AccumulatorUse::kWrite, OperandType::kReg)                      \
                                                                             \
/* Perform a stack guard check */                                            \
V(StackCheck, AccumulatorUse::kNone)                                         \
                                                                             \
/* Update the pending message */                                             \
V(SetPendingMessage, AccumulatorUse::kReadWrite)                             \
                                                                             \
/* Non-local flow control */                                                 \
V(Throw, AccumulatorUse::kRead)                                              \
V(ReThrow, AccumulatorUse::kRead)                                            \
V(Return, AccumulatorUse::kRead)                                             \
V(ThrowReferenceErrorIfHole, AccumulatorUse::kRead, OperandType::kIdx)       \
V(ThrowSuperNotCalledIfHole, AccumulatorUse::kRead)                          \
V(ThrowSuperAlreadyCalledIfNotHole, AccumulatorUse::kRead)                   \
```

谢 谢

欢迎交流合作

2020/2/13