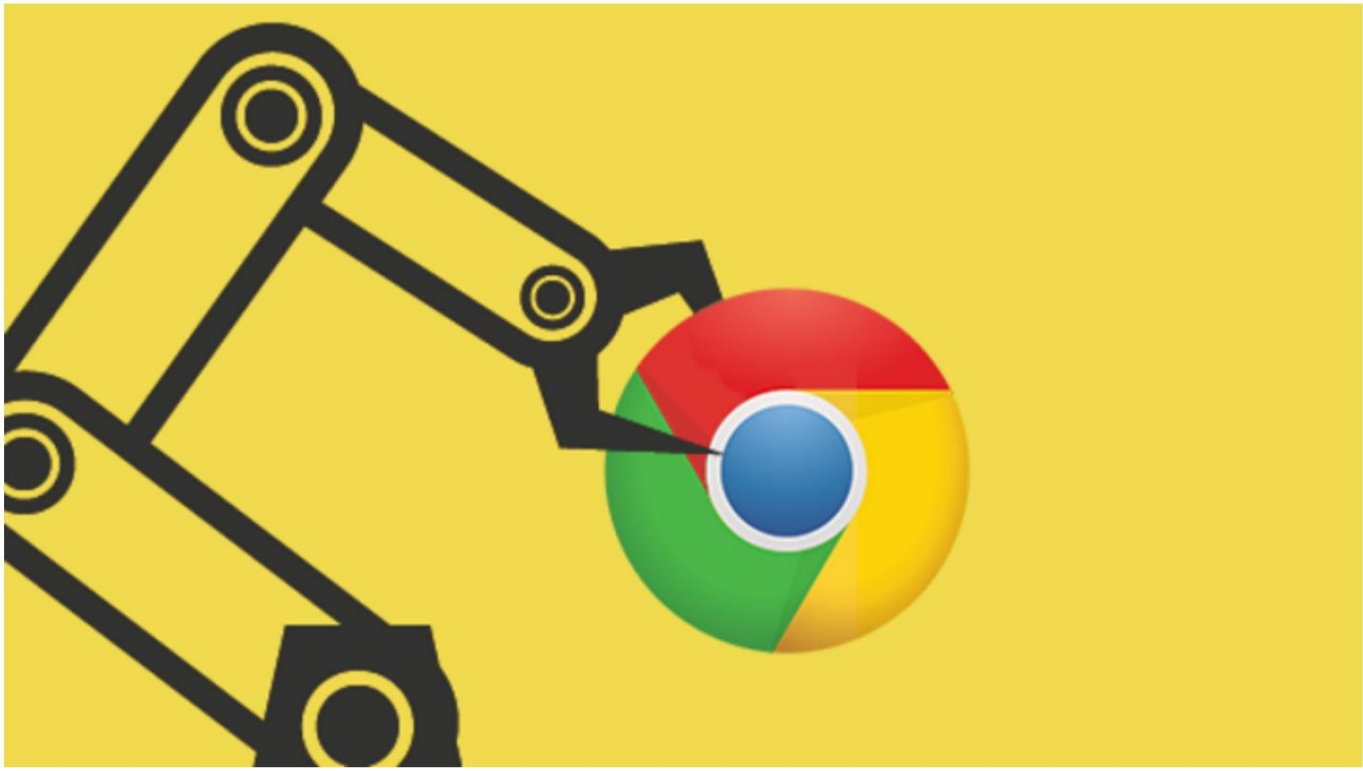
"Chrome V8 Source Code" 36. String.prototype.concat source code analysis



1 Introduction

Strings are an important data type in JavaScript. Their importance is not only reflected in the fact that strings are the most widely used data type, but also in the fact that V8 uses a large number of technical means to modify and optimize string operations. The next few articles will focus on the related operations of strings. This article first explains the source code and related data structures of String.prototype.concat, and then demonstrates the calling, loading and execution process of

String.prototype.concat through test cases. **Note** (1) Sea of Nodes is the leading knowledge of this article. Please refer to Cliff's 1993 paper From Quads to Graphs. (2) The environment used in this article is: V8 7.9, win10 x64, VS2019.

2 String.prototype.concat source code

The test case code is as follows:

```
var txt1 = "he ", txt2="is ", txt3="HuiDou ", txt4="."; var bio = txt1.concat(txt2,txt3,txt4);
console.log(bio);
```

concet() is implemented using TF_BUILTIN. The function name of concet() in V8 is StringPrototypeConcat, and the number is 888. The source code is as follows:

```
1. TF_BUILTIN(StringPrototypeConcat, CodeStubAssembler) {
2. ca_.Goto(&block0, torque_arguments.frame, torque_arguments.base,
torque_arguments.length, parameter0, parameter1);
3.         if (block0.is_used()) {//Omit.....
4. ca_.Bind(&block0, &tmp0, &tmp1, &tmp2, &tmp3, &tmp4);
5.             ca_.SetSourcePosition("../../../src/builtins/string.tq", 128);
6.             compiler::TNode<String> tmp5;
7.             USE(tmp5);
8.             tmp5 = FromConstexpr6String18ATconstexpr_string_156(state_,
"String.prototype.concat");
9.             compiler::TNode<String> tmp6;
10.              USE(tmp6);
11.              tmp6 = CodeStubAssembler(state_).ToThisString(compiler::TNode<Context>
{tmp3}, compiler::TNode<Object>{tmp4}, compiler::TNode<String>{tmp5});
12.              ca_.SetSourcePosition("../../../src/builtins/string.tq", 131);
13.              compiler::TNode<IntPtrT> tmp7;
14.              USE(tmp7);
15.              tmp7 = Convert8ATintptr8ATintptr_1494(state_, compiler::TNode<IntPtrT>
{tmp2});
16.              ca_.SetSourcePosition("../../../src/builtins/string.tq", 132);
17.              compiler::TNode<IntPtrT> tmp8;
18.              USE(tmp8);
19.              tmp8 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 0);
20.              ca_.Goto(&block3, tmp0, tmp1, tmp2, tmp3, tmp4, tmp6, tmp7, tmp8);
twenty one.         }
twenty two.         if (block3.is_used()) {//Omitted.....
twenty three.             ca_.Bind(&block3, &tmp9, &tmp10, &tmp11, &tmp12, &tmp13, &tmp14, &tmp15,
&tmp16);
twenty four.             compiler::TNode<BoolT> tmp17;
25.              USE(tmp17);
26.              tmp17 = CodeStubAssembler(state_).IntPtrLessThan(compiler::TNode<IntPtrT>
{tmp16}, compiler::TNode<IntPtrT>{tmp15});
27.              ca_.Branch(tmp17, &block1, &block2, tmp9, tmp10, tmp11, tmp12, tmp13,
tmp14, tmp15, tmp16);
28.         }
29.         if (block1.is_used()) {//Omit.....
30.             ca_.Bind(&block1, &tmp18, &tmp19, &tmp20, &tmp21, &tmp22, &tmp23, &tmp24,
&tmp25);
31.             ca_.SetSourcePosition("../../../src/builtins/string.tq", 133);
32.             compiler::TNode<Object> tmp26;
33.             USE(tmp26);
34.             tmp26 =
CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<R
awPtrT>{tmp18}, compiler::TNode<RawPtrT>{tmp19}, compiler::TNode<IntPtrT>{tmp20}},
compiler::TNode<IntPtrT>{tmp25});
35. compiler::TNode<String> tmp27;
36.             USE(tmp27);
37.             tmp27 =
CodeStubAssembler(state_).ToString_Inline(compiler::TNode<Context>{tmp21},
compiler::TNode<Object>{tmp26});
38.             ca_.SetSourcePosition("../../../src/builtins/string.tq", 134);
39.             compiler::TNode<String> tmp28;
40.             USE(tmp28);
```

```
41.             tmp28 = StringAdd_82(state_, compiler::TNode<Context>{tmp21},
compiler::TNode<String>{tmp23}, compiler::TNode<String>{tmp27});
42.             ca_.SetSourcePosition("../../../src/builtins/string.tq", 132);
43.             ca_.Goto(&block4, tmp18, tmp19, tmp20, tmp21, tmp22, tmp28, tmp24,
tmp25);
44.         }
45.         if (block4.is_used()) {//Omitted.....
46.             ca_.Bind(&block4, &tmp29, &tmp30, &tmp31, &tmp32, &tmp33, &tmp34, &tmp35,
&tmp36);
47.             compiler::TNode<IntPtrT> tmp37;
48.             USE(tmp37);
49.             tmp37 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 1);
50.             compiler::TNode<IntPtrT> tmp38;
51.             USE(tmp38);
52.             tmp38 = CodeStubAssembler(state_).IntPtrAdd(compiler::TNode<IntPtrT>
{tmp36}, compiler::TNode<IntPtrT>{tmp37});
53.             ca_.Goto(&block3, tmp29, tmp30, tmp31, tmp32, tmp33, tmp34, tmp35,
tmp38);
54.         }
55.         if (block2.is_used()) {//Omit.....
56.             ca_.Bind(&block2, &tmp39, &tmp40, &tmp41, &tmp42, &tmp43, &tmp44, &tmp45,
&tmp46);
57.             ca_.SetSourcePosition("../../../src/builtins/string.tq", 136);
58.             arguments.PopAndReturn(tmp44);
59.         }
60. }
```

Four blocks (block0-block4) are defined in the above code, and their functions are:

block0 creates three variables: the initial string tmp6 (that is, "he" in the test case), the total number of splices tmp7 (the number in the test case is 3) and the number of completed splicing tmp8 (initial value is 0);

block1 concatenates two strings together to generate a new string;

block2 returns the final result;

block3 determines whether the number of completed splicing is less than the total number of splicings. If it is less, jump to block1, if not, jump to blcok2;

block4 adds 1 to the number of completed splices.

It can be seen from the distribution position of these blocks in the code that they use a while loop to implement string splicing. Figure 1 gives The location of StringPrototypeConcat source code.

Figure 1

The following explains the important functions used by StringPrototypeConcat:

(1) ToThisString (line 11) is used to convert objects into strings. The source code is as follows:

```
1. TNode<String> CodeStubAssembler::ToThisString(/*omitted*/) {
2. BIND(&if_valueisnotsmi);//Omit....
3. { TNode<Uint16T> value_instance_type = LoadInstanceType(CAST(value));
4.         Label if_valueisnotstring(this, Label::kDeferred);
5.         Branch(IsStringInstanceType(value_instance_type), &if_valueisstring,
6.                        &if_valueisnotstring);
7.          BIND(&if_valueisnotstring);
8.          { Label if_valueisnullorundefined(this, Label::kDeferred);
9.             GotoIf(IsNullOrUndefined(value), &if_valueisnullorundefined);
10.              var_value.Bind(CallBuiltin(Builtins::kToString, context, value));
11.              Goto(&if_valueisstring);
12.              BIND(&if_valueisnullorundefined);
13.              {ThrowTypeError(context, MessageTemplate::kCalledOnNullOrUndefined,
14.                                          method_name);}
15.          }
16.       }
17. BIND(&if_valueissmi);
18. {var_value.Bind(CallBuiltin(Builtins::kNumberToString, context, value));
19.          Goto(&if_valueisstring); }
20. BIND(&if_valueisstring);
21. return CAST(var_value.value());
```

Lines 2-16 in the above code are used to convert non-small integer data into strings. Non-small integer can be arrays, floating point numbers, etc. That The specific function of the conversion operation is implemented by kToString on line 10; lines 17-19 are used to convert small integer data into strings. The specific function of the replacement operation is implemented by kNumberToString on line 18. Subsequent articles will explain kToString and kNumberToString separately. method.

(2) IntPtrAdd (line 52) implements the addition of integer data. The source code is as follows:

```
1. TNode<WordT> CodeAssembler::IntPtrAdd(SloppyTNode<WordT> left,
2.                                       SloppyTNode<WordT> right) {
3. intptr_t left_constant;
4. bool is_left_constant = ToIntPtrConstant(left, &left_constant);
5. intptr_t right_constant;
6. bool is_right_constant = ToIntPtrConstant(right, &right_constant);
7. if (is_left_constant) {
8.         if (is_right_constant) {
9.             return IntPtrConstant(left_constant + right_constant);}
10.            if (left_constant == 0) { return right;}
11. } else if (is_right_constant) {
12.            if (right_constant == 0) {return left;}
13.            }
14. return UncheckedCast<WordT>(raw_assembler()->IntPtrAdd(left, right));}
```

Lines 7-13 in the above code are used for constant addition operations. Lines 7-8 determine the left and right values of the addition. If they are both constants, the results are calculated directly.

(Line 9); The 10th line of code determines the lvalue, and if it is zero, it directly returns the rvalue; the 14th line of code indicates that when both the left and right values are variables, then Compute nodes need to be added.

(3) The Builtins::kStringAdd_CheckNone method is called in StringAdd_82 (line 41), which uses CodeStubAssembler::StringAdd completes the addition operation of strings. The source code of StringAdd is as follows:

```
1. TNode<String> CodeStubAssembler::StringAdd(Node* context, TNode<String> left,
2.                                            TNode<String> right) {
3. TNode<Uint32T> left_length = LoadStringLengthAsWord32(left);
4. GotoIfNot(Word32Equal(left_length, Uint32Constant(0)), &check_right);
5. result = right;
6. Goto(&done_native);
7. BIND(&check_right);
8. TNode<Uint32T> right_length = LoadStringLengthAsWord32(right);
9. GotoIfNot(Word32Equal(right_length, Uint32Constant(0)), &cons);
10. result = left;
11. Goto(&done_native);
12. BIND(&cons);
13.        {
14.            TNode<Uint32T> new_length = Uint32Add(left_length, right_length);
15.            GotoIf(Uint32GreaterThan(new_length, Uint32Constant(String::kMaxLength)),
16.                   &runtime);
17.            TVARIABLE(String, var_left, left);
18.            TVARIABLE(String, var_right, right);
19.            Variable* input_vars[2] = {&var_left, &var_right};
20.            Label non_cons(this, 2, input_vars);
twenty one.    Label slow(this, Label::kDeferred);
twenty two.    GotoIf(Uint32LessThan(new_length, Uint32Constant(ConsString::kMinLength)),
twenty three.          &non_cons);
twenty four.   result =AllocateConsString(new_length, var_left.value(),
var_right.value());
25.            Goto(&done_native);
26.            BIND(&non_cons);
27.            TNode<Int32T> left_instance_type = LoadInstanceType(var_left.value());
28.            TNode<Int32T> right_instance_type = LoadInstanceType(var_right.value());
```

```
29.         TNode<Int32T> ored_instance_types =
30.                 Word32Or(left_instance_type, right_instance_type);
31.         TNode<Word32T> xored_instance_types =
32.                 Word32Xor(left_instance_type, right_instance_type);
33.         GotoIf(IsSetWord32(xored_instance_types, kStringEncodingMask), &runtime);
34.         GotoIf(IsSetWord32(ored_instance_types, kStringRepresentationMask),
&slow);
35.         TNode<IntPtrT> word_left_length = Signed(ChangeUint32ToWord(left_length));
36.         TNode<IntPtrT> word_right_length =
Signed(ChangeUint32ToWord(right_length));
37.         Label two_byte(this);
38.         GotoIf(Word32Equal(Word32And(ored_instance_types,
39.                                                 Int32Constant(kStringEncodingMask)),
40.                                         Int32Constant(kTwoByteStringTag)),&two_byte);
41.         result = AllocateSeqOneByteString(new_length);
42.         CopyStringCharacters(/*copy left string*/);
43.         CopyStringCharacters(/*Copy the right string*/);
44.         Goto(&done_native);
45.         BIND(&two_byte);
46.         {
47.             result = AllocateSeqTwoByteString(new_length);
48.             CopyStringCharacters(/*copy left string*/);
49.             CopyStringCharacters(/*Copy the right string*/);
50.             Goto(&done_native); }
51.         BIND(&slow);
52.         {
53.             MaybeDerefIndirectStrings(&var_left, left_instance_type, &var_right,
54.                                                 right_instance_type, &non_cons);
55.             Goto(&runtime);}}
56. BIND(&runtime);
57. {//Omit...
58.         Goto(&done); }
59. BIND(&done_native);
60. { Goto(&done); }
61. BIND(&done);
62.         return result.value();}
```

In the above code, lines 3-6 determine whether the length of the lvalue is zero. If it is zero, the rvalue is returned directly;

Lines 7-11 of the code determine whether the length of the rvalue is zero. If it is zero, the lvalue is returned directly;

Lines 14-15 of the code determine whether the length of the new string is greater than the maximum length of the string specified by V8. If it is greater, use the runtime method.

reason;

Line 22 of the code determines whether the length of the new string is less than the minimum length of the string specified by V8. If it is less, use the slow or runtime method.

type processing;

Lines 27-34 of code determine whether the encoding of the left and right values are consistent and both are sequential strings. If the result is true, lines 35-50 are executed.

code;

Lines 35-50 of code use different methods to create and return new strings based on single and double bytes, and the function is executed;

Lines 51-58 of code use slow and runtime methods to process strings.

Figure 2 shows the call stack for StringAdd().

Figure 2

```
7532    }
7533
7534    TNode<String> CodeStubAssembler::StringAdd(Node* context, TNode<String> left,
7535                                               TNode<String> right) {
7536      TVARIABLE(String, result); 已用时间 <= 1ms
7537      Label check_right(this), runtime(this, Label::kDeferred), cons(this),
7538          done(this, &result), done_native(this, &result);
7539      Counters* counters = isolate()->counters();
7540
7541      TNode<Uint32T> left_length = LoadStringLengthAsWord32(left);
7542      GotoIfNot(Word32Equal(left_length, Uint32Constant(0)), &check_right);
7543      result = right;
7544      Goto(&done_native);
```

88 % 未找到相关问题

调用堆栈

名称	语言
v8.dll!v8::internal::CodeStubAssembler::StringAdd(v8::internal::compiler::Node * context, v8::internal::compiler::TNode<v8::internal::String> left, v8::internal::compiler::T...	C++
v8.dll!v8::internal::StringAdd_CheckNoneAssembler::GenerateStringAdd_CheckNoneImpl() 行 296	C++
v8.dll!v8::internal::Builtins::Generate_StringAdd_CheckNone(v8::internal::compiler::CodeAssemblerState * state) 行 292	C++
v8.dll!v8::internal::`anonymous namespace'::BuildWithCodeStubAssemblerCS(v8::internal::Isolate * isolate, int builtin_index, void(*)(v8::internal::compiler::CodeAssembl...	C++
v8.dll!v8::internal::SetupIsolateDelegate::SetupBuiltinsInternal(v8::internal::Isolate * isolate) 行 325	C++
v8.dll!v8::internal::SetupIsolateDelegate::SetupBuiltins(v8::internal::Isolate * isolate) 行 20	C++
v8.dll!v8::internal::Isolate::Init(v8::internal::ReadOnlyDeserializer * read_only_deserializer, v8::internal::StartupDeserializer * startup_deserializer) 行 3445	C++
v8.dll!v8::internal::Isolate::InitWithoutSnapshot() 行 3308	C++
v8.dll!v8::Isolate::Initialize(v8::Isolate * isolate, const v8::Isolate::CreateParams & params) 行 8094	C++
v8.dll!v8::Isolate::New(const v8::Isolate::CreateParams & params) 行 8106	C++
d8.exe!v8::Shell::Main(int argc, char * * argv) 行 3514	C++
d8.exe!main(int argc, char * * argv) 行 3640	C++
[外部代码]	

3.String.prototype.concat test

The bytecode of the test case is as follows:

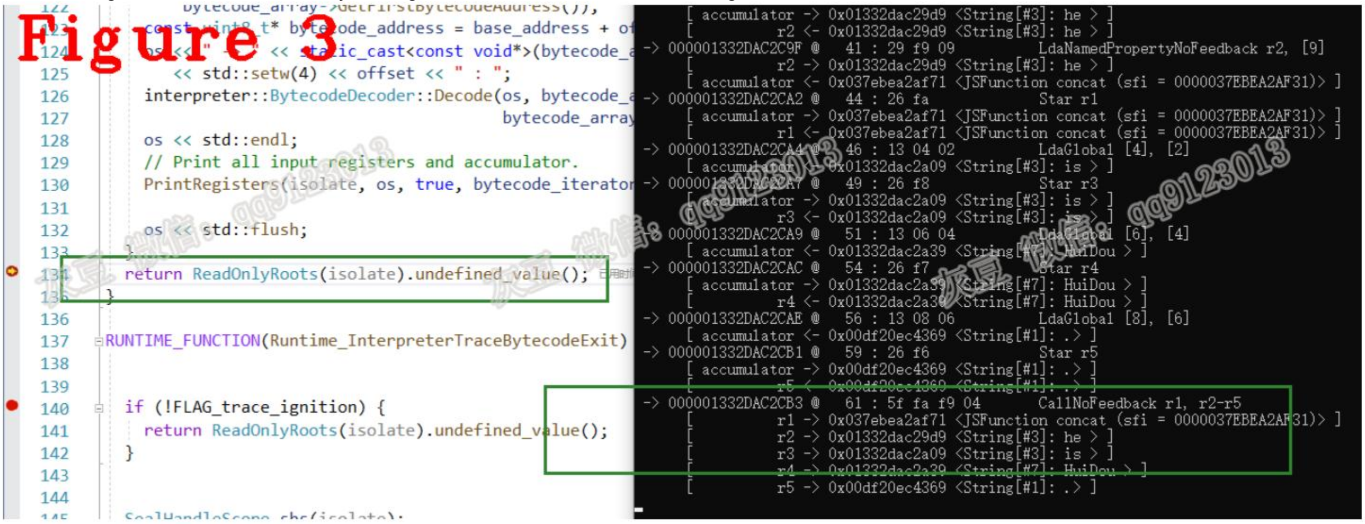
```
1. //Omit.....
2.          11 S> 000001332DAC2C86 @ 16 : 12 01 11 E>          LdaConstant [1]
            000001332DAC2C88 @ 18 : 15 02 0a 3. 4. //Omit...   StaGlobal [2], [10]

5.          64 S> 000001332DAC2C9A @ 36 : 13 02 00             LdaGlobal [2], [0]
6.                000001332DAC2C9D @ 39 : 26 f9 69 E>          Star r2
7.          000001332DAC2C9F @ 41 : 29 f9 09
LdaNamedPropertyNoFeedback r2, [9]
8. //Omit.....
9.          69 E> 000001332DAC2CB3 @ 61 : 5f fa f9 04 10. //Omit...   CallNoFeedback r1, r2-r5

11. Constant pool (size = 13)
12. 000001332DAC2BC9: [FixedArray] in OldSpace
13.       - map: 0x00df20ec0169 <Map>
14.       - length: 13
15.                  0: 0x01332dac2b09 <FixedArray[20]>
16.                  1: 0x01332dac29d9 <String[#3]: he >
17.                  2: 0x01332dac29c1 <String[#4]: txt1>
18.                  3: 0x01332dac2a09 <String[#3]: is >
19.                  4: 0x01332dac29f1 <String[#4]: txt2>
20.                  5: 0x01332dac2a39 <String[#7]: HuiDou>
twenty one.          6: 0x01332dac2a21 <String[#4]: txt3>
twenty two.          7: 0x00df20ec4369 <String[#1]: .>
twenty three.        8: 0x01332dac2a51 <String[#4]: txt4>
```

```
twenty four.  9: 0x037ebea28b49 <String[#6]: concat>
25.  10: 0x01332dac2a69 <String[#3]: bio>
26.  11: 0x037ebea336f1 <String[#7]: console>
27.  12: 0x037ebea32d31 <String[#3]: log>
```

In the above code, lines 2-3 load and store the string "he"; lines 5-6 save the string "he" to the r2 register; line 7 loads concat method; line 9 calls the concat method, the value of the r1 register is the address of concat, r2-r5 are "he", "is", "HuiDou" in order and".". Debug test method: Start assembly tracking from CallNoFeedback, as shown in Figure 3.



Technical summary

(1) Before splicing, it is necessary to determine whether the type, encoding, and single and double bytes of the left and right strings are consistent;

(2) Splicing uses a circular method to splice strings into pairs;

(3) The maximum length of a string is String::kMaxLength (1073741799).

Okay, that's it for today, see you next time.

Personal abilities are limited and there are shortcomings and mistakes. Criticisms and corrections are welcome.

WeChat: qq9123013 Note: v8 communication email: v8blink@outlook.com