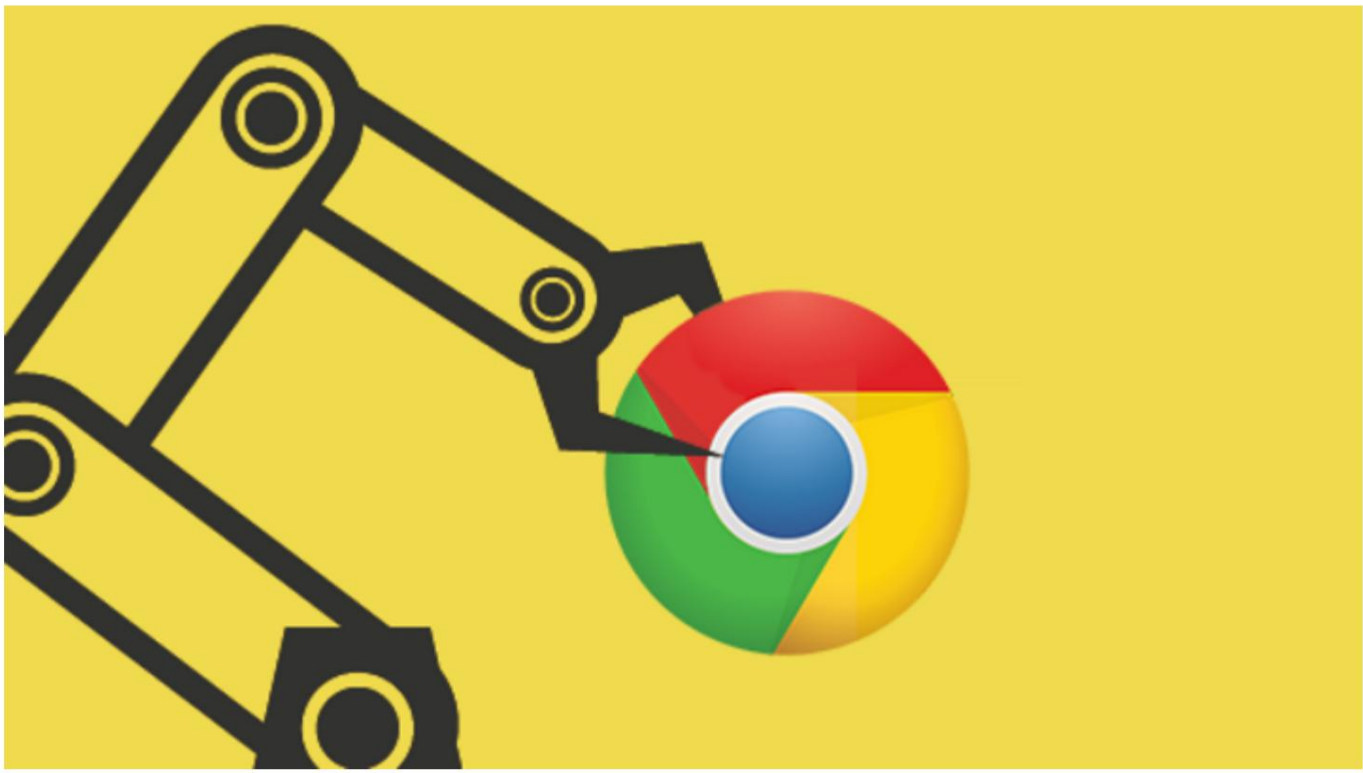
"Chrome V8 Source Code" 28. Analysis of substring source code and implicit conventions



1 abstract

This article is the fourth of the Builtin topic, mainly analyzing the source code of substring. There are two implementation methods for substring. One uses CSA implementation.

Now, another implementation uses Runtime. This article explains the substring method implemented by CSA and V8's implicit convention on string length and type.

2 CSA implementation of substring

When extracting a substring between two specified subscripts in a string, V8 gives priority to using the substring method implemented by CSA. The source code is as follows:

```
1. TF_BUILTIN(StringPrototypeSubstring, CodeStubAssembler) {
2.         if (block0.is_used()) {//Many generations are omitted
code.....................................
3.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 33);
4.             tmp5 = FromConstexpr6String18ATconstexpr_string_156(state_,
"String.prototype.substring");
5.             tmp6 = CodeStubAssembler(state_).ToThisString(compiler::TNode<Context>
{tmp3}, compiler::TNode<Object>{tmp4}, compiler::TNode<String>{tmp5});
6.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 34);
7.             tmp7 =
CodeStubAssembler(state_).LoadStringLengthAsSmi(compiler::TNode<String>{tmp6});
8.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 37);
9. tmp8 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 0);
10.             tmp9 =
CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<R
```

```
awPtrT>{tmp0}, compiler::TNode<RawPtrT>{tmp1}, compiler::TNode<IntPtrT>{tmp2}},
compiler::TNode<IntPtrT>{tmp8});
11.             tmp10 = ToSmiBetweenZeroAnd_343(state_, compiler::TNode<Context>{tmp3},
compiler::TNode<Object>{tmp9}, compiler::TNode<Smi>{tmp7});
12.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
13. tmp11 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 1);
14.             tmp12 =
CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<R
awPtrT>{tmp0}, compiler::TNode<RawPtrT>{tmp1}, compiler::TNode<IntPtrT>{tmp2}},
compiler::TNode<IntPtrT>{tmp11});
15.             tmp13 = Undefined_64(state_);
16.             tmp14 = CodeStubAssembler(state_).TaggedEqual(compiler::TNode<Object>
{tmp12}, compiler::TNode<HeapObject>{tmp13});
17. ca_.Branch(tmp14, &block1, &block2, tmp0, tmp1, tmp2, tmp3, tmp4, tmp6,
tmp7, tmp10);
18.         }
19. if (block1.is_used()) {
20.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 41);
twenty one.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
twenty two.             ca_.Goto(&block4, tmp15, tmp16, tmp17, tmp18, tmp19, tmp20, tmp21, tmp22,
tmp21);
twenty three. }
twenty four.         if (block2.is_used()) {
25.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 42);
26.             tmp31 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 1);
27.             tmp32 =
CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<R
awPtrT>{tmp23}, compiler::TNode<RawPtrT>{tmp24}, compiler::TNode<IntPtrT>{tmp25}},
compiler::TNode<IntPtrT>{tmp31});
28.             tmp33 = ToSmiBetweenZeroAnd_343(state_, compiler::TNode<Context>{tmp26},
compiler::TNode<Object>{tmp32}, compiler::TNode<Smi>{tmp29});
29.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
30. ca_.Goto(&block3, tmp23, tmp24, tmp25, tmp26, tmp27, tmp28, tmp29, tmp30,
tmp33);
31.         }
32.         if (block4.is_used()) {
33.             ca_.Goto(&block3, tmp34, tmp35, tmp36, tmp37, tmp38, tmp39, tmp40, tmp41,
tmp42);
34.         }
35.         if (block3.is_used()) {
36.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 43);
37.             tmp52 = CodeStubAssembler(state_).SmiLessThan(compiler::TNode<Smi>
{tmp51}, compiler::TNode<Smi>{tmp50});
38.             ca_.Branch(tmp52, &block5, &block6, tmp43, tmp44, tmp45, tmp46, tmp47,
tmp48, tmp49, tmp50, tmp51);
39. }
40.         if (block5.is_used()) {
41.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 44);
42.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 45);
43.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 46);
44.             ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 43);
45.             ca_.Goto(&block6, tmp53, tmp54, tmp55, tmp56, tmp57, tmp58, tmp59, tmp61,
tmp60);
46.         }
```

```
47.         if (block6.is_used()) {
48.           ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 48);
49.           tmp71 = CodeStubAssembler(state_).SmiUntag(compiler::TNode<Smi>{tmp69});
50.           tmp72 = CodeStubAssembler(state_).SmiUntag(compiler::TNode<Smi>{tmp70});
51.           tmp73 = CodeStubAssembler(state_).SubString(compiler::TNode<String>
{tmp67}, compiler::TNode<IntPtrT>{tmp71}, compiler::TNode<IntPtrT>{tmp72});
52.           arguments.PopAndReturn(tmp73);
53.         }
54. }
```

The above code is generated by string-substring.tq instructing the compiler, and its location is in the V8\v8\src\out\default\gen\torque-generated\src\builtins directory, which means that it is generated during the compilation of V8.

(1) The third line of code sets the source code. The source code comes from line 33 of the string-substring.tq file, see Figure 1;

(2) codeStubAssembler(state_).ToThisString() (line 5 of code) converts this into a string; line 6 of code sets the source Code, see Figure 1; CodeStubAssembler(state_).LoadStringLengthAsSmi() (line 7 of code) calculates the string length, parameter

The value of the number tmp6 is the execution result of the 5th line of code. Since the coding style of lines 6 and 7 is the same as that of lines 3 and 5, we can pass

Line-by-line analysis of string-substring.tq understands CodeStubAssembler.

Figure 1

```
27
28
29    // ES6 #sec-string.prototype.substring
30    transitioning javascript builtin StringPrototypeSubstring(
31        js-implicit context: Context, receiver: JSAny)(...arguments): String {
32      // Check that {receiver} is coercible to Object and convert it to a String.
33      const string: String = ToThisString(receiver, 'String.prototype.substring');
34      const length = string.length_smi;
35
36      // Conversion and bounds-checks for {start}.
37      let start: Smi = ToSmiBetweenZeroAnd(arguments[0], length);
38
39      // Conversion and bounds-checks for {end}.
40      let end: Smi = arguments[1] == Undefined ?
41          length :
42          ToSmiBetweenZeroAnd(arguments[1], length);
43      if (end < start) {
44        const tmp: Smi = end;
45        end = start;
46        start = tmp;
47      }
48      return SubString(string, SmiUntag(start), SmiUntag(end));
49    }
50  }
51
```

The following describes other key functions in the substring source code:

(1) ca_.Goto() jumps to the label position. Its first parameter is the label. The source code is as follows:

```
template <class... T, class... Args>
void Goto(CodeAssemblerParameterizedLabel<T...>* label, Args... args) {
  label->AddInputs(args...);
  Goto(label->plain_label());
}
```

(2) ca_.Bind() sets the label, the source code is as follows:

```
template <class... T>
void Bind(CodeAssemblerParameterizedLabel<T...>* label, TNode<T>*... phis) {
  Bind(label->plain_label());
  label->CreatePhis(phis...);
}
```

(3) ca_.Branch() branch jump, the source code is as follows:

```
template <class... T, class... Args>
void Branch(TNode<BoolT> condition,
            CodeAssemblerParameterizedLabel<T...>* if_true,
            CodeAssemblerParameterizedLabel<T...>* if_false, Args... args) {
  if_true->AddInputs(args...);
  if_false->AddInputs(args...);
  Branch(condition, if_true->plain_label(), if_false->plain_label());
}
```

The parameter condition is the condition, and the parameters if_true and if_false are jump labels.

(4) LoadStringLengthAsSmi() and SmiUntag() are member methods of CodeStubAssembler. Summarize The functions of TF_BUILTIN(StringPrototypeSubstring, CodeStubAssembler) are as follows:

(1) Convert this to a string and get the length;

(2) Determine whether the length of substring (sublen) is less than length;

(3) Call CodeStubAssembler.SubString to complete the substring operation.

The source code of CodeStubAssembler.SubString is as follows:

```
1. TNode<String> CodeStubAssembler::SubString(TNode<String> string,
2.                                            TNode<IntPtrT> from,
3.                                            TNode<IntPtrT> to) {
4. //Omit a lot
5.     Label external_string(this);
6.     {
7.       if (FLAG_string_slices) {
8.         Label next(this);
9.         GotoIf(IntPtrLessThan(substr_length,
10.                              IntPtrConstant(SlicedString::kMinLength)),
11.                &next);
12.          Counters* counters = isolate()->counters();
13.          IncrementCounter(counters->sub_string_native(), 1);
14.          Label one_byte_slice(this), two_byte_slice(this);
15.          Branch(IsOneByteStringInstanceType(to_direct.instance_type()),
16.                 &one_byte_slice, &two_byte_slice);
17.          BIND(&one_byte_slice);
18.          {
19.            var_result = AllocateSlicedOneByteString(
```

```
                    Unsigned(TruncateIntPtrToInt32(substr_length)), direct_string,
                    SmiTag(offset));
                Goto(&end);
            }
            BIND(&two_byte_slice);
            {
                var_result = AllocateSlicedTwoByteString(
                    Unsigned(TruncateIntPtrToInt32(substr_length)), direct_string,
                    SmiTag(offset));
                Goto(&end);
            }
            BIND(&next);
        }
        GotoIf(to_direct.is_external(), &external_string);
        var_result = AllocAndCopyStringCharacters(direct_string, instance_type,
                                                  offset, substr_length);
        Counters* counters = isolate()->counters();
        IncrementCounter(counters->sub_string_native(), 1);
        Goto(&end);
    }
    BIND(&external_string);
    {
        TNode<RawPtrT> const fake_sequential_string =
            to_direct.PointerToString(&runtime);
        var_result = AllocAndCopyStringCharacters(
            fake_sequential_string, instance_type, offset, substr_length);
        Counters* counters = isolate()->counters();
        IncrementCounter(counters->sub_string_native(), 1);
        Goto(&end);
    }
    BIND(&empty);
    {
    }
    BIND(&single_char);
    {
        TNode<Int32T> char_code = StringCharCodeAt(string, from);
        var_result = StringFromSingleCharCode(char_code);
        Goto(&end);
    }
    BIND(&original_string_or_invalid_length);
    {
//Omit a lot
    }
    BIND(&runtime);
    {
        var_result =
            CAST(CallRuntime(Runtime::kStringSubstring, NoContextConstant(),
string,
                             SmiTag(from), SmiTag(to)));
        Goto(&end);
    }
    BIND(&end);
    return var_result.value();
}
```

FLAG_string_slices (line 7 above) is the enable flag of slices. It is defined in flag-definitions.h. The source code is as follows:

```
// Flags for data representation optimizations
DEFINE_BOOL_READONLY(string_slices, true, "use string slices")
```

Line 9 of code GotoIf() calculates the value of substr_length and jumps to the label next if it is less than 13.

Line 15 of code Branch() determines whether a string is a single-byte character or a double-byte character.

Lines 17-23 and 24-30 handle single-byte and double-byte situations respectively, which will be explained later.

Lines 40-49 of code BIND(&external_string) operate external strings. External strings refer to strings that are not in the V8 heap, such as from
The string referenced in the DOM is the external string. Use the Runtime method when operating external strings.

Lines 53-58 of code: When sublength=1, call StringCharCodeAt() to complete the corresponding operation and return the result.

Lines 63-70 of code: When the string is an external string, call Runtime_StringSubstring to complete the corresponding operation and return the result.

In V8, when slice generates a new string, if the new string length is greater than SlicedString::kMinLength, it will not apply for new memory, but use
The start pointer and end pointer refer to the original string. Taking single-byte string characters as an example to explain the slice method, the source code is as follows:

```
1. TNode<String> CodeStubAssembler::AllocateSlicedOneByteString(
2.             TNode<Uint32T> length, TNode<String> parent, TNode<Smi> offset) {
3.           return AllocateSlicedString(RootIndex::kSlicedOneByteStringMap, length,
4.                                                                 parent, offset);
5. }
6. //Separator line.....................
7. TNode<String> CodeStubAssembler::AllocateSlicedString(RootIndex
map_root_index,
8.                                                                                 TNode<Uint32T> length,
9.                                                                                 TNode<String> parent,
10.                                                                                  TNode<Smi> offset) {
11.          DCHECK(map_root_index == RootIndex::kSlicedOneByteStringMap ||
12.                 map_root_index == RootIndex::kSlicedStringMap);
13.          TNode<HeapObject> result = Allocate(SlicedString::kSize);
14.          DCHECK(RootsTable::IsImmortalImmovable(map_root_index));
15.          StoreMapNoWriteBarrier(result, map_root_index);
16.          StoreObjectFieldNoWriteBarrier(result, SlicedString::kHashFieldOffset,
17.                                                 Int32Constant(String::kEmptyHashField),
18.                                                 MachineRepresentation::kWord32);
19.          StoreObjectFieldNoWriteBarrier(result, SlicedString::kLengthOffset, length,
20.                                                 MachineRepresentation::kWord32);
twenty one.  StoreObjectFieldNoWriteBarrier(result, SlicedString::kParentOffset, parent,
twenty two.                                        MachineRepresentation::kTagged);
twenty three. StoreObjectFieldNoWriteBarrier(result, SlicedString::kOffsetOffset, offset,
twenty four.                                       MachineRepresentation::kTagged);
25.          return CAST(result);
26. }
```

In the above code, AllocateSlicedOneByteString() is the entry function, which calls the AllocateSlicedString() function. Line 13
The code creates a SlicedString object (result); lines 16-24 of the code store the sublength, parent string base address and offset in the result.
Slicing is complete.

Technical

summary (1) string-substring.tq is the Builtin source code handwritten by the developer, string-substring-tq-csa.cc and .h are the Builtin source code generated by Tq;

(2) The value of SlicedString::kMinLength is 13, news =substring(start,stop), when the length of news is less than 13, the copy mechanism is used, and when the length is greater than 13, the reference

mechanism is used; **(3)** Because the Runtime_substring method is used, the operation efficiency of external strings is low. Okay, that's it for today, see you next time.

Personal abilities are limited, there are shortcomings and mistakes,

criticisms and corrections are welcome WeChat: qq9123013 Note: v8 Communication Zhihu: https://www.zhihu.com/people/v8blink