《Chrome V8源码》28.分析substring源码和隐式约定

1 摘要

本篇文章是Builtin专题的第四篇，主要分析`substring`的源码。`substring`有两种实现方法，一种采用CSA实现，另一种采用Runtime实现。本文讲解CSA实现的`substring`方法以及V8对字符串长度和类型的隐式约定。

2 substring的CSA实现

提取字符串中介于两个指定下标之间的子字符串时，V8优先使用CSA实现的substring方法，源码如下：

```
TF_BUILTIN(StringPrototypeSubstring, CodeStubAssembler) {
  if (block0.is_used()) {//省略了很多代码.......................................
    ca_.SetSourcePosition("../../../../src/builtins/string-substring.tq", 33);
    tmp5 = FromConstexpr6String18ATconstexpr_string_156(state_, "String.prototype.substring");
    tmp6 = CodeStubAssembler(state_).ToThisString(compiler::TNode<Context>{tmp3}, compiler::TNode<Object>{tmp4}, compiler::TNode<String>{tmp5});
    ca_.SetSourcePosition("../../../../src/builtins/string-substring.tq", 34);
    tmp7 = CodeStubAssembler(state_).LoadStringLengthAsSmi(compiler::TNode<String>{tmp6});
    ca_.SetSourcePosition("../../../../src/builtins/string-substring.tq", 37);
    tmp8 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 0);
    tmp9 = CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<R
awPtrT>{tmp0}, compiler::TNode<RawPtrT>{tmp1}, compiler::TNode<IntPtrT>{tmp2}}, compiler::TNode<IntPtrT>{tmp8});
   tmp10 = ToSmiBetweenZeroAnd_343(state_, compiler::TNode<Context>{tmp3}, compiler::TNode<Object>{tmp9}, compiler::TNode<Smi>{tmp7});
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
   tmp11 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 1);
   tmp12 = CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<RawPtrT>{tmp0}, compiler::TNode<RawPtrT>{tmp1}, compiler::TNode<IntPtrT>{tmp2}}, compiler::TNode<IntPtrT>{tmp11});
   tmp13 = Undefined_64(state_);
   tmp14 = CodeStubAssembler(state_).TaggedEqual(compiler::TNode<Object>{tmp12}, compiler::TNode<HeapObject>{tmp13});
   ca_.Branch(tmp14, &block1, &block2, tmp0, tmp1, tmp2, tmp3, tmp4, tmp6, tmp7, tmp10);
  }
  if (block1.is_used()) {
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 41);
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
   ca_.Goto(&block4, tmp15, tmp16, tmp17, tmp18, tmp19, tmp20, tmp21, tmp22, tmp21);
  }
  if (block2.is_used()) {
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 42);
   tmp31 = FromConstexpr8ATintptr17ATconstexpr_int31_150(state_, 1);
   tmp32 = CodeStubAssembler(state_).GetArgumentValue(TorqueStructArguments{compiler::TNode<RawPtrT>{tmp23}, compiler::TNode<RawPtrT>{tmp24}, compiler::TNode<IntPtrT>{tmp25}}, compiler::TNode<IntPtrT>{tmp31});
   tmp33 = ToSmiBetweenZeroAnd_343(state_, compiler::TNode<Context>{tmp26}, compiler::TNode<Object>{tmp32}, compiler::TNode<Smi>{tmp29});
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 40);
   ca_.Goto(&block3, tmp23, tmp24, tmp25, tmp26, tmp27, tmp28, tmp29, tmp30, tmp33);
  }
  if (block4.is_used()) {
   ca_.Goto(&block3, tmp34, tmp35, tmp36, tmp37, tmp38, tmp39, tmp40, tmp41, tmp42);
  }
  if (block3.is_used()) {
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 43);
   tmp52 = CodeStubAssembler(state_).SmiLessThan(compiler::TNode<Smi>{tmp51}, compiler::TNode<Smi>{tmp50});
   ca_.Branch(tmp52, &block5, &block6, tmp43, tmp44, tmp45, tmp46, tmp47, tmp48, tmp49, tmp50, tmp51);
  }
  if (block5.is_used()) {
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 44);
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 45);
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 46);
   ca_.SetSourcePosition("../../../src/builtins/string-substring.tq", 43);
   ca_.Goto(&block6, tmp53, tmp54, tmp55, tmp56, tmp57, tmp58, tmp59, tmp61, tmp60);
  }
```

```
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.  }
```

上述代码由`string-substring.tq`指导编译器生成，其位置在`V8\v8\src\out\default\gen\torque-generated\src\builtins`目录下，这意味它在编译V8过程中生成。
（1） 第3行代码设置源码，源码来自`string-substring.tq`文件的第33行，见图1；
（2） `codeStubAssembler(state_).ToThisString()`（第5行代码）把this转成字串符； 第6行代码设置源码，见图1；`CodeStubAssembler(state_).LoadStringLengthAsSmi()`（第7行代码）计算字符串长度，参数tmp6的值是第5行代码的执行结果。由于第6、7行代码与第3、5行的编码风格一样，所以可以通过对`string-substring.tq`的逐行分析看懂CodeStubAssembler。

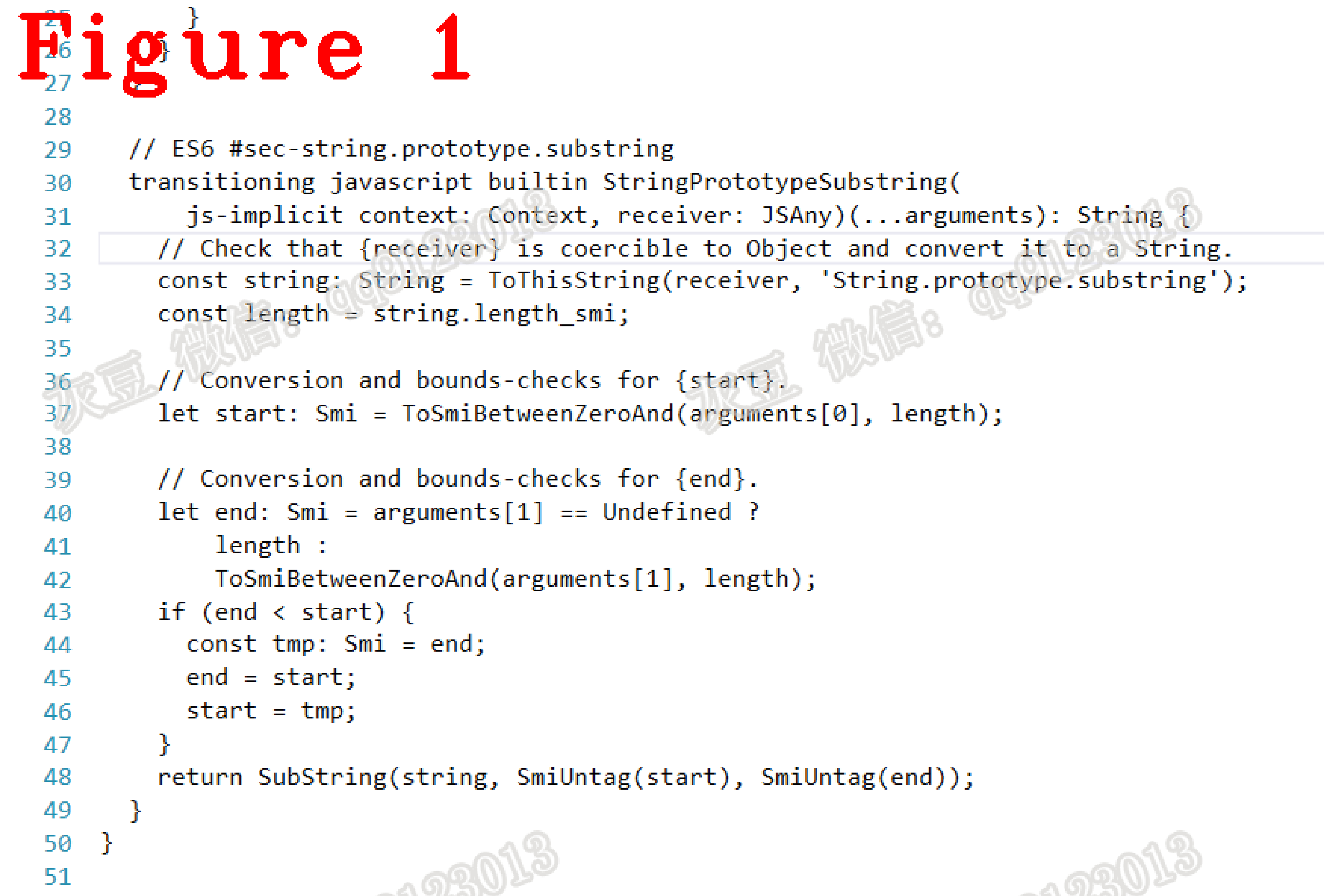

Figure 1

```
25      }
26    }
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
```

下面说明substring源码中的其它关键功能：
（1） `ca_.Goto()`跳转到标签位置，它的第一参数是标签，源码如下：

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

(2) ca_.Bind()设置标签，源码如下：

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

(3) ca_.Branch()分支跳转，源码如下：

```
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());
}
```

其中参数condition是条件，参数if_true、if_false是跳转标签。
(4) LoadStringLengthAsSmi()和SmiUntag()是CodeStubAssembler的成员方法。 总结TF_BUILTIN(StringPrototypeSubstring, CodeStubAssembler)的功能为如下三点：
(1) 把this转换为字符串并获取长度length；
(2) 判断substring的长度（sublen）是否小于length；
(3) 调用CodeStubAssembler.SubString完成substring操作。
CodeStubAssembler.SubString的源码如下：

```
TNode<String> CodeStubAssembler::SubString(TNode<String> string,
                                          TNode<IntPtrT> from,
                                          TNode<IntPtrT> to) {
//省略很多
  Label external_string(this);
  {
    if (FLAG_string_slices) {
      Label next(this);
      GotoIf(IntPtrLessThan(substr_length,
                             IntPtrConstant(SlicedString::kMinLength)),
              &next);
       Counters* counters = isolate()->counters();
       IncrementCounter(counters->sub_string_native(), 1);
       Label one_byte_slice(this), two_byte_slice(this);
       Branch(IsOneByteStringInstanceType(to_direct.instance_type()),
              &one_byte_slice, &two_byte_slice);
       BIND(&one_byte_slice);
       {
         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);
  {
//省略很多
  }
  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（上述第7行代码）是切片的使能标记，它定义在flag-definitions.h中，源码如下：

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

第9行代码GotoIf()计算substr_length的值，如果小于13则跳转到标签next。
第15行代码Branch()判断字符串是单字节字符还是双字节字符。
第17-23行、24-30行分别处理单字节、双字节两种情况，稍后讲解。
第40-49行代码BIND(&external_string)操作外部字符串，外部字符串指的是不在V8 heap中的字符串，如从DOM中引用的字符串就是外部字符串。操作外部字符串时使用Runtime方法。
第53-58行代码：当sublength=1时，调用StringCharCodeAt()完成相应的操作并返回结果。
第63-70行代码：当字符串为外部字符串时，调用Runtime_StringSubstring完成相应的操作并返回结果。
在V8中，slice生成新字符串时，如果新字符串长度大于SlicedString::kMinLength则不申请新内存，而是使用开始指针和结束指针引用原字符串。以单字节字串符为例讲解slice方法，源码如下：

```
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.  //分隔线..............................
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);
21.    StoreObjectFieldNoWriteBarrier(result, SlicedString::kParentOffset, parent,
22.                                   MachineRepresentation::kTagged);
23.    StoreObjectFieldNoWriteBarrier(result, SlicedString::kOffsetOffset, offset,
24.                                   MachineRepresentation::kTagged);
25.    return CAST(result);
26.  }
```

上述代码中AllocateSlicedOneByteString()是入口函数，调用AllocateSlicedString()函数。第13行代码创建SlicedString对象（result）；第16-24行代码把sublength、父亲字符串基址和偏移量存入result中，slice完毕。

技术总结
（1） string-substring.tq是开发者手写的Builtin源码，string-substring-tq-csa.cc和.h是Tq生成的Builtin源码；
（2） SlicedString::kMinLength的值是13，news=substring(start,stop)，news的长度小于13时用copy机制，大于13时用引用机制；
（3） 因为使用了Runtime_substring方法，所以外部字符串的操作效率低。
好了，今天到这里，下次见。

个人能力有限，有不足与纰漏，欢迎批评指正
微信：qq9123013 备注：v8交流 知乎：https://www.zhihu.com/people/v8blink

本文由灰豆原创发布
转载，请参考转载声明，注明出处： https://www.anquanke.com/post/id/260386
安全客 - 有思想的安全新媒体