《Chrome V8 源码》40.Runtime substring详解

1 介绍

Runtime 是一系列采用 C++ 语言编写的功能方法，它实现了大量 JavaScript 运行期间需要的 native 功能，例如 String add、String split。接下来这几篇文章将介绍一些 Runtime 方法。本文分析 Runtime_StringSubstring 方法的源码和重要数据结构，讲解 Runtime_StringSubstring 方法的触发条件。
注意 Runtime 方法的加载、调用以及 RUNTIME_FUNCTION 宏模板请参见第十六篇文章。

2 Runtime_StringSubstring 源码分析

源码如下：

```
1.  RUNTIME_FUNCTION(Runtime_StringSubstring) {
2.    HandleScope scope(isolate);
3.    DCHECK_EQ(3, args.length());
4.    CONVERT_ARG_HANDLE_CHECKED(String, string, 0);
5.    CONVERT_INT32_ARG_CHECKED(start, 1);
6.    CONVERT_INT32_ARG_CHECKED(end, 2);
7.    DCHECK_LE(0, start);
8.    DCHECK_LE(start, end);
9.    DCHECK_LE(end, string->length());
10.    isolate->counters()->sub_string_runtime()->Increment();
11.    return *isolate->factory()->NewSubString(string, start, end);
12.  }
```

上述代码中，第 3 行代码检测参数的数量是否为 3，不是就报错；
第 4-6 行代码分别获取 string、start、end 三个参数；
第 7-9 行代码分别检测 string 的长度是否小于 0、end 的值是否小于 start、string的长度是否小于 end，任何一个结果为真时报错；
第 11 行代码 NewSubString(string, start, end) 函数内部调用了 NewProperSubString(str, begin, end) 方法，其源码如下：

```
1.  Handle<String> Factory::NewProperSubString(Handle<String> str, int begin,
2.                                             int end) {
3.    DCHECK(begin > 0 || end < str->length());
4.    str = String::Flatten(isolate(), str);
5.    int length = end - begin;
6.    if (length <= 0) return empty_string();
7.    if (length == 1) {
8.      return LookupSingleCharacterStringFromCode(str->Get(begin));  }
9.    if (length == 2) {
10.     uint16_t c1 = str->Get(begin);
11.     uint16_t c2 = str->Get(begin + 1);
12.     return MakeOrFindTwoCharacterString(isolate(), c1, c2);  }
13.   if (!FLAG_string_slices || length < SlicedString::kMinLength) {
14.     if (str->IsOneByteRepresentation()) {
15.       Handle<SeqOneByteString> result =
16.           NewRawOneByteString(length).ToHandleChecked();
17.       DisallowHeapAllocation no_gc;
18.       uint8_t* dest = result->GetChars(no_gc);
19.       String::WriteToFlat(*str, dest, begin, end);
20.       return result;
21.     } else {
22.       Handle<SeqTwoByteString> result =
23.           NewRawTwoByteString(length).ToHandleChecked();
24.       DisallowHeapAllocation no_gc;
25.       uc16* dest = result->GetChars(no_gc);
26.       String::WriteToFlat(*str, dest, begin, end);
27.       return result;  }
28.   }
29.   int offset = begin;
30.   if (str->IsSlicedString()) {
31.     Handle<SlicedString> slice = Handle<SlicedString>::cast(str);
32.     str = Handle<String>(slice->parent(), isolate());
33.     offset += slice->offset();  }
34.   if (str->IsThinString()) {
35.     Handle<ThinString> thin = Handle<ThinString>::cast(str);
36.     str = handle(thin->actual(), isolate());  }
37.   DCHECK(str->IsSeqString() || str->IsExternalString());
38.   Handle<Map> map = str->IsOneByteRepresentation()
39.                         ? sliced_one_byte_string_map()
40.                         : sliced_string_map();
41.   Handle<SlicedString> slice(
42.       SlicedString::cast(New(map, AllocationType::kYoung)), isolate());
43.   slice->set_hash_field(String::kEmptyHashField);
44.   slice->set_length(length);
45.   slice->set_parent(*str);
```

```
46.     slice->set_offset(offset);
47.     return slice;
48.   }
```

上述代码中，第 4 行代码对 string 进行 Flatten 处理，稍后讲解；
第 5-6 行代码判断是否返回空字符串；
第 7-8 行代码处理 length 值为 1 的情况，也就是 substring(1)；
第 9-12 行代码处理 length 值为 2 的情况，也就是 substring(2)；
第 13 行代码 !FLAG_string_slices 即不允许切片，例如在 result = example.substring(x,y)；这条语句中，不能采用 **offset** 和 **length** 的方式表示 result；length < SlicedString::kMinLength 说明当 result 的长度小于切片的最小限制时不采用切片方式表示 result;
第 14-19 行代码处理单字节字符串，申请 length 长度的堆空间（dest），然后复制 begin 至 end 之间的字符到 dest 中;
第 22-26 行代码处理双字节字符串，方式同上；
重点说明： “result = example.substring(x,y)” 中 example 是 slice 或 Thin 类型时的 substring 处理方法：
（1） slice 类型：采用 **offset** 和 **length** 获取父字符串（parent）中部分切片的字符串
第 30-31 行代码检测出 example 为 slice 类型后，将其转换为 slice 类型字符串；
第 32 行代码获得 slice 字符串的父亲字符串（str）；
第 33 行代码 slice 字符串在 str 中的起始位置（offset）加上 begin 得到 substring 在 str 中的超始位置；
第 41-47 行代码申请新的切片字符串堆空间，将该堆空间的父指针指向父亲（str），再设置新的offset 和 length，返回结果；
（2） Thin 类型：直接引用另一个字符串对象的字符串
第 34-36 行代码检测出 example 为 Thin 类型后，将转换为 Thin 类型字符串；获得被引用的字符串；
第 41-47 行代码同上。
下面说明 NewProperSubString 中用到的重要方法：
（1） Flatten 方法，源码如下：

```
1.  Handle<String> String::Flatten(Isolate* isolate, Handle<String> string,
2.                                 AllocationType allocation) {
3.  if (string->IsConsString()) {
4.    Handle<ConsString> cons = Handle<ConsString>::cast(string);
5.    if (cons->IsFlat()) {
6.      string = handle(cons->first(), isolate);
7.    } else {
8.      return SlowFlatten(isolate, cons, allocation); } }
9.   if (string->IsThinString()) {
10.     string = handle(Handle<ThinString>::cast(string)->actual(), isolate);
11.     DCHECK(!string->IsConsString()); }
12.  return string;}
```

上述代码将 ConsString 和 ThinString 转换为连续存储的字符串。第 8 行代码 SlowFlatten 方法利用循环将 ConsString 的两个子串组合成一个连续的字符串，其源码如下：

```
1.  Handle<String> String::SlowFlatten(Isolate* isolate, Handle<ConsString> cons,
2.                                     AllocationType allocation) {
3.    while (cons->first().length() == 0) {
```

```
4.      if (cons->second().IsConsString() && !cons->second().IsFlat()) {
5.        cons = handle(ConsString::cast(cons->second()), isolate);
6.      } else {
7.        return String::Flatten(isolate, handle(cons->second(), isolate));
8.      }
9.    }
10.    int length = cons->length();
11.    allocation =
12.        ObjectInYoungGeneration(*cons) ? allocation : AllocationType::kOld;
13.    Handle<SeqString> result;
14.    if (cons->IsOneByteRepresentation()) {
15.      Handle<SeqOneByteString> flat =
16.          isolate->factory()
17.              ->NewRawOneByteString(length, allocation)
18.              .ToHandleChecked();
19.      DisallowHeapAllocation no_gc;
20.      WriteToFlat(*cons, flat->GetChars(no_gc), 0, length);
21.      result = flat;
22.    } else {
23.      Handle<SeqTwoByteString> flat =
24.          isolate->factory()
25.              ->NewRawTwoByteString(length, allocation)
26.              .ToHandleChecked();
27.      DisallowHeapAllocation no_gc;
28.      WriteToFlat(*cons, flat->GetChars(no_gc), 0, length);
29.      result = flat;
30.    }
31.    cons->set_first(*result);
32.    cons->set_second(ReadOnlyRoots(isolate).empty_string());
33.    DCHECK(result->IsFlat());
34.    return result;
35.  }
```

上述代码中，第 3-7 行代码使用 while 分别处理 ConsString 的两个子串。因为子串也可以是 ConsString 类型，所以可能再次调用 Flatten 方法；
第 10-34 行代码分别处理单字节、双字节类型的 ConsString。处理方式是申请堆空间，并拷贝 ConsString 的两个字符串到刚申请的堆空间中，返回结果。

3 Runtime_StringSubstring 触发条件

V8 官方文档说明了 --allow-natives-syntax 和 %-prefix 可以唤起 Runtime 方法，但我们要学习的是在什么样的情况下 V8 会使用 Runtime 方法来处理 JavaScript 源码。我们从 CodeStubAssembler::SubString() 入手分析，因为该方法是 V8 最先调用的方法，当该方法失败时才会用 Runtime 方法。该方法的详细讲解参见第 28 篇文章。下面给出 CodeStubAssembler::SubString() 的部分源码：

```
1.  TNode<String> CodeStubAssembler::SubString(TNode<String> string,
2.                                             TNode<IntPtrT> from,
3.                                             TNode<IntPtrT> to) {
4.    Label original_string_or_invalid_length(this);
```

```
5.    GotoIf(UintPtrGreaterThanOrEqual(substr_length, string_length),
6.           &original_string_or_invalid_length);
7.    TNode<String> direct_string = to_direct.TryToDirect(&runtime);
8.    TNode<IntPtrT> offset = IntPtrAdd(from, to_direct.offset());
9.    TNode<Int32T> const instance_type = to_direct.instance_type();
10.    BIND(&original_string_or_invalid_length);
11.    {//省略很......
12.      CSA_ASSERT(this, IntPtrEqual(substr_length, string_length));
13.      GotoIf(UintPtrGreaterThan(from, IntPtrConstant(0)), &runtime);
14.    }
15.    BIND(&runtime);
16.    {
17.      var_result =
18.          CAST(CallRuntime(Runtime::kStringSubstring, NoContextConstant(),
string,
19.                           SmiTag(from), SmiTag(to)));
20.      Goto(&end);
21.    }}
```

上述代码中，第 7 行代码把字符串转换为直接字符串失败时会调用 Runtime 方法；第 10 行代码满足 original_string_or_invalid_length 条件时也会调用 Runtime 方法；另外，外部字符串也使用 Runtime 方法处理（代码中省略）。我们构造的测试用例如下：

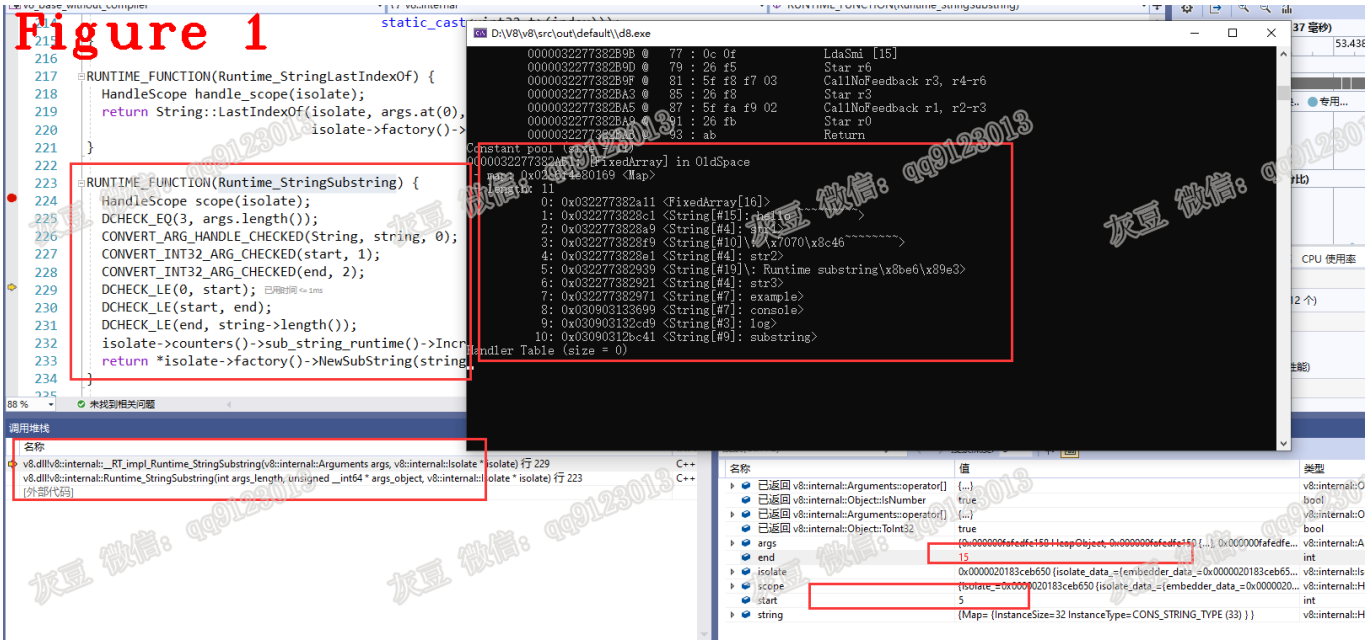
```
var str1="hello ~~~~~~~~~~";
var str2="灰豆~~~~~~~~";
var str3="Runtime substring详解";
var example = str1+str2+str3;
console.log(example.substring(5,15));
```

该代码中，example 是 ConsString 类型，它的两个子串分别是 ConsString 和 双字节类型。example 的 TryToDirect 转换失败导致触发 Runtime 方法。下面给出测试用例代码的常量池数据：

```
Constant pool (size = 11)
0000032277382AB1: [FixedArray] in OldSpace
 - map: 0x02c6f4e80169 <Map>
 - length: 11
           0: 0x032277382a11 <FixedArray[16]>
           1: 0x0322773828c1 <String[#15]: hello ~~~~~~~~~~>
           2: 0x0322773828a9 <String[#4]: str1>
           3: 0x0322773828f9 <String[#10]\: \x7070\x8c46~~~~~~~~>
           4: 0x0322773828e1 <String[#4]: str2>
           5: 0x032277382939 <String[#19]\: Runtime substring\x8be6\x89e3>
           6: 0x032277382921 <String[#4]: str3>
           7: 0x032277382971 <String[#7]: example>
           8: 0x030903133699 <String[#7]: console>
           9: 0x030903132cd9 <String[#3]: log>
          10: 0x03090312bc41 <String[#9]: substring>
Handler Table (size = 0)
```

我们可以看到 str1 是单字节字符串，str2 和 str3 是双字节字符串。

图 1 给出了 Runtime_StringSubstring 的调用堆栈。



图中可以看到 start 和end 的值分别是 5 和 15。

技术总结

(1) SeqString，V8 堆内的连续存储的字符串，分为 OneByte、TwoByte 两类；

(2) ConsString，采用指针形式表示的拼接（first + second）字符串；

(3) SliceString，采用 offset 和 length 表达父字符串（parent）部分切片内容的字符串；

(4) ThinString，直接引用另一个字符串对象的字符串；

(5) ExternalString， V8 堆空间之外的字符串。

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 邮箱：v8blink@outlook.com

本文由灰豆原创发布

转载出处： https://www.anquanke.com/post/id/264196

安全客 - 有思想的安全新媒体