

《Chrome V8 源码》 39. String.prototype.split 源码分析



1 介绍

字符串是 JavaScript 中的重要数据类型，其重要性不仅体现在字符串是应用最多最广泛的数据类型，更体现在 V8 中使用了大量的技术手段来修饰和优化字符串的操作。接下来的几篇文章将集中讲解字符串的相关操作。本文先讲解 `String.prototype.split` 的源码以及相关数据结构，再通过测试用例演示 `String.prototype.split` 的调用、加载和执行过程。

注意 (1) Sea of Nodes 是本文的先导知识，请参考 Cliff 1993 年发表的论文 *From Quads to Graphs*。(2) 本文所用环境为：V8 7.9、win10 x64、VS2019。

2 String.prototype.split 源码

测试用例代码如下：

```
var str="How9are9you9doing9today?";
var n=str.split("9");
console.log(n);
```

`split()` 采用 `TF_BUILTIN` 实现，`replace()` 在 V8 中的函数名是 `StringPrototypeSplit`，编号是 596，源码如下：

```
1. TF_BUILTIN(StringPrototypeSplit, StringBuiltinsAssembler) {
2.   const int kSeparatorArg = 0;
```

```

3.     const int kLimitArg = 1;
4.     TNode<IntPtrT> const argc =
5.         ChangeInt32ToIntPtr(Parameter(Descriptor::kJSActualArgumentsCount));
6.     CodeStubArguments args(this, argc);
7.     TNode<Object> receiver = args.GetReceiver();
8.     TNode<Object> const separator =
args.GetOptionalArgumentValue(kSeparatorArg);
9.     TNode<Object> const limit = args.GetOptionalArgumentValue(kLimitArg);
10.    TNode<NativeContext> context = CAST(Parameter(Descriptor::kContext));
11.    TNode<Smi> smi_zero = SmiConstant(0);
12.    RequireObjectCoercible(context, receiver, "String.prototype.split");
13.    MaybeCallFunctionAtSymbol(/*省略...*/);
14.    TNode<String> subject_string = ToString_Inline(context, receiver);
15.    TNode<Number> limit_number = Select<Number>(
16.        IsUndefined(limit), [=] { return NumberConstant(kMaxUInt32); },
17.        [=] { return ToUint32(context, limit); });
18.    TNode<String> const separator_string = ToString_Inline(context, separator);
19.    Label return_empty_array(this);
20.    GotoIf(TaggedEqual(limit_number, smi_zero), &return_empty_array);
21.    {
22.        Label next(this);
23.        GotoIfNot(IsUndefined(separator), &next);
24.        const ElementsKind kind = PACKED_ELEMENTS;
25.        TNode<NativeContext> const native_context = LoadNativeContext(context);
26.        TNode<Map> array_map = LoadJSArrayElementsMap(kind, native_context);
27.        TNode<Smi> length = SmiConstant(1);
28.        TNode<IntPtrT> capacity = IntPtrConstant(1);
29.        TNode<JSArray> result = AllocateJSArray(kind, array_map, capacity,
length);
30.        TNode<FixedArray> fixed_array = CAST(LoadElements(result));
31.        StoreFixedArrayElement(fixed_array, 0, subject_string);
32.        args.PopAndReturn(result);
33.        BIND(&next);
34.    }
35.    {
36.        Label next(this);
37.        GotoIfNot(SmiEqual(LoadStringLengthAsSmi(separator_string), smi_zero),
38.            &next);
39.        TNode<Smi> subject_length = LoadStringLengthAsSmi(subject_string);
40.        GotoIf(SmiEqual(subject_length, smi_zero), &return_empty_array);
41.        args.PopAndReturn(
42.            StringToArray(context, subject_string, subject_length,
limit_number));
43.        BIND(&next);
44.    }
45.    TNode<Object> const result =
46.        CallRuntime(Runtime::kStringSplit, context, subject_string,
47.            separator_string, limit_number);
48.    args.PopAndReturn(result);
49.    BIND(&return_empty_array);
50.    {
51.        const ElementsKind kind = PACKED_ELEMENTS;
52.        TNode<NativeContext> const native_context = LoadNativeContext(context);
53.        TNode<Map> array_map = LoadJSArrayElementsMap(kind, native_context);

```

```

54.     TNode<Smi> length = smi_zero;
55.     TNode<IntPtrT> capacity = IntPtrConstant(0);
56.     TNode<JSArray> result = AllocateJSArray(kind, array_map, capacity,
length);
57.     args.PopAndReturn(result);
58. }
59. }

```

上述代码中，第 7 行代码 receiver 代表测试用例中的字符串 str；

第 8 行代码 separator 代表测试用例中的 "9"；

第 9 行代码读取 limit；

第 13 行代码如果 separator 是正则表达式，就使用 MaybeCallFunctionAtSymbol() 完成 split 操作；

第 14 行代码把 receiver 转换为字符串，并保存到 subject_string 中；

第 15 行代码把 limit 的类型转换为 Number；

第 16 行代码把 separator 转换为字符串，并保存到 separator_string 中；

第 20 行代码判断 limit 是否等于零，等于返回空字符串，split 退出；

第 21 行代码判断 separator 是否等于 Undefined，不等于则跳转到第 33 行代码；

第 24-32 行代码实现 ECMA 的规定：“if {separator} is undefined, the result should be an array of size 1 containing the entire string.”；

第 37 行代码判断 separator 的长度是否等于零，不等于则跳转到第 43 行代码；

第 39-41 行代码用于实现 separator="" 时的 split 功能，即创建由 receiver 中的所有字符组成的数组；

第 45 行代码当 “separator 长度大于零” 和 “limit 未定义或大于零” 两个条件同时满足时，该行用于实现 split 功能；

第 49-59 行代码返回空数组。

下面说明 StringPrototypeSplit 中使用的重要函数：

(1) StringToArray 方法，在本文中该方法实现了 separator="" 时的 split 功能，源码如下：

```

1.  TNode<JSArray> StringBuiltinsAssembler::StringToArray(
2.      TNode<NativeContext> context, TNode<String> subject_string,
3.      TNode<Smi> subject_length, TNode<Number> limit_number) {
4.      TVARIABLE(JSArray, result_array);
5.      TNode<Uint16T> instance_type = LoadInstanceType(subject_string);
6.      GotoIfNot(IsOneByteStringInstanceType(instance_type), &call_runtime);
7.      {
8.          TNode<Smi> length_smi =
9.              Select<Smi>(TaggedIsSmi(limit_number),
10.                 [=] { return SmiMin(CAST(limit_number), subject_length);
11.                 [=] { return subject_length; });
12.          TNode<IntPtrT> length = SmiToIntPtr(length_smi);
13.          ToDirectStringAssembler to_direct(state(), subject_string);
14.          to_direct.TryToDirect(&call_runtime);
15.          TNode<FixedArray> elements = CAST(AllocateFixedArray(
16.              PACKED_ELEMENTS, length,
AllocationFlag::kAllowLargeObjectAllocation));
17.          TNode<RawPtrT> string_data =
18.              to_direct.PointerToData(&fill_thehole_and_call_runtime);
19.          TNode<IntPtrT> string_data_offset = to_direct.offset();
20.          TNode<FixedArray> cache = SingleCharacterStringCacheConstant();

```

```

21.     BuildFastLoop(
22.         IntPtrConstant(0), length,
23.         [&](Node* index) {
24.             CSA_ASSERT(this, WordEqual(to_direct.PointerToData(&call_runtime),
25.                                     string_data));
26.             TNode<Int32T> char_code =
27.                 UncheckedCast<Int32T>(Load(MachineType::Uint8(), string_data,
28.                                     IntPtrAdd(index,
string_data_offset)));
29.             TNode<UIntPtrT> code_index = ChangeUInt32ToWord(char_code);
30.             TNode<Object> entry = LoadFixedArrayElement(cache, code_index);
31.             GotoIf(IsUndefined(entry), &fill_thehole_and_call_runtime);
32.             StoreFixedArrayElement(elements, index, entry);
33.         },
34.         1, ParameterMode::INTPTR_PARAMETERS, IndexAdvanceMode::kPost);
35.     TNode<Map> array_map = LoadJSArrayElementsMap(PACKED_ELEMENTS, context);
36.     result_array = AllocateJSArray(array_map, elements, length_smi);
37.     Goto(&done);
38.     BIND(&fill_thehole_and_call_runtime);
39.     {
40.         FillFixedArrayWithValue(PACKED_ELEMENTS, elements, IntPtrConstant(0),
41.                                 length, RootIndex::kTheHoleValue);
42.         Goto(&call_runtime); } }
43.     BIND(&call_runtime);
44.     {
45.         result_array = CAST(CallRuntime(Runtime::kStringToArray, context,
46.                                     subject_string, limit_number));
47.         Goto(&done);}
48.     BIND(&done);
49.     return result_array.value();}

```

上述代码中，第 6 行代码判断 subject_string 是否为单字节字符串，如果不是则跳转到第 43 行；

第 8 行代码计算数组长度，如果定义了 limit，数组长度取 subject_string.length 和 limit 之中的最小值；如果未定义，数组长度等于 subject_string.length；

第 14 行代码将间接字符串转换为直接字符串，转换失败时调用 runtime 处理；

第 15 行代码分配数组 elements；

第 21-34 行代码使用 BuildFastLoop 方式填充 elements，填充失败时使用 runtime 处理；**注意**BuildFastLoop 使用了 Turbofan 优化技术，请自行学习。

第 40-45 行代码使用 runtime 方式生成数组；

(2) Runtime_StringSplit 源码如下：

```

1.  RUNTIME_FUNCTION(Runtime_StringSplit) {
2.  int subject_length = subject->length();
3.  int pattern_length = pattern->length();
4.  CHECK_LT(0, pattern_length);
5.  if (limit == 0xFFFFFFFFu) { // 省略..... }
6.  subject = String::Flatten(isolate, subject);
7.  pattern = String::Flatten(isolate, pattern);
8.  std::vector<int>* indices = GetRewoundRegexIndicesList(isolate);
9.  FindStringIndicesDispatch(isolate, *subject, *pattern, indices, limit);

```

```
10.  if (static_cast<uint32_t>(indices->size()) < limit) {
11.      indices->push_back(subject_length);}
12.  int part_count = static_cast<int>(indices->size());
13.  Handle<JSArray> result =
14.      isolate->factory()->NewJSArray(PACKED_ELEMENTS, part_count, part_count,
15.                                      INITIALIZE_ARRAY_ELEMENTS_WITH_HOLE);
16.  DCHECK(result->HasObjectElements());
17.  Handle<FixedArray> elements(FixedArray::cast(result->elements()), isolate);
18.  if (part_count == 1 && indices->at(0) == subject_length) {
19.      elements->set(0, *subject);
20.  } else {
21.      int part_start = 0;
22.      FOR_WITH_HANDLE_SCOPE(isolate, int, i = 0, i, i < part_count, i++, {
23.          int part_end = indices->at(i);
24.          Handle<String> substring =
25.              isolate->factory()->NewProperSubString(subject, part_start,
part_end);
26.          elements->set(i, *substring);
27.          part_start = part_end + pattern_length;
28.      });}
29.  if (limit == 0xFFFFFFFFu) {//省略.....}
30.  TruncateRegexpIndicesList(isolate);
31.  return *result;
```

上述代码中，第 2 行代码 subject 表示测试用例字符串 str;

第 3 行代码 pattern 表示 separator;

第 9 行代码使用 pattern 切分 subject，并将切分结果保存在 indices 中。切分结果是切片的数量以及每个切片的开始、结尾下标。

第 13 行代码申请数组 elements，该数组长度等于切片数量；

第 19 行代码切片数量为 1 时，把 subject 复制到 elements 中；

第 20-28 行代码使用循环把每个切片复制到 elements 中；

图 1 给出了初始化 StringPrototypeSplit 时的调用堆栈。



2 String.prototype.split 测试

测试代码的字节码如下：

```

1. 000000A369D42A96 @ 16 : 12 01 LdaConstant [1]
2. 000000A369D42A98 @ 18 : 15 02 04 StaGlobal [2], [4]
3. 000000A369D42A9B @ 21 : 13 02 00 LdaGlobal [2], [0]
4. 000000A369D42A9E @ 24 : 26 f9 Star r2
5. 000000A369D42AA0 @ 26 : 29 f9 03 LdaNamedPropertyNoFeedback r2,
[3]
6. 000000A369D42AA3 @ 29 : 26 fa Star r1
7. 000000A369D42AA5 @ 31 : 12 04 LdaConstant [4]
8. 000000A369D42AA7 @ 33 : 26 f8 Star r3
9. 000000A369D42AA9 @ 35 : 5f fa f9 02 CallNoFeedback r1, r2-r3
10. 000000A369D42AAD @ 39 : 15 05 06 StaGlobal [5], [6]
11. 000000A369D42AB0 @ 42 : 13 06 08 LdaGlobal [6], [8]
12. 000000A369D42AB3 @ 45 : 26 f9 Star r2
13. 000000A369D42AB5 @ 47 : 29 f9 07 LdaNamedPropertyNoFeedback r2,
[7]
14. 000000A369D42AB8 @ 50 : 26 fa Star r1
15. 000000A369D42ABA @ 52 : 13 05 02 LdaGlobal [5], [2]
16. 000000A369D42ABD @ 55 : 26 f8 Star r3
17. 000000A369D42ABF @ 57 : 5f fa f9 02 CallNoFeedback r1, r2-r3
18. 000000A369D42AC3 @ 61 : 26 fb Star r0
19. 000000A369D42AC5 @ 63 : ab Return
20. Constant pool (size = 8)
21. 000000A369D42A01: [FixedArray] in OldSpace
22. - map: 0x03abc3a00169 <Map>
23. - length: 8

```

```
24. 0: 0x00a369d429a1 <FixedArray[8]>
25. 1: 0x00a369d428c1 <String[#24]: How9are9you9doing9today?>
26. 2: 0x00a369d428a9 <String[#3]: str>
27. 3: 0x0233195eba31 <String[#5]: split>
28. 4: 0x00a369d42901 <String[#1]: 9>
29. 5: 0x00a369d428e9 <String[#1]: n>
30. 6: 0x0233195f3699 <String[#7]: console>
31. 7: 0x0233195f2cd9 <String[#3]: log>
```

上述代码中，第 1-4 行代码读取字符串 “How9are9you9doing9today?” 并保存到 r2 寄存器中；

第 5-6 行代码获取字符串方法 split 并保存到 r1 寄存器中；

第 7-8 行代码获取 separator 字符串并保存到 r3 寄存器中。测试用例的 separator 是 ‘9’。

第 9 行代码 CallNoFeedback 调用 split 方法，并传递 r2、r3 两个参数给 split 方法。

技术总结

- (1) 多数情况下，split 方法由 runtime 实现；
- (2) v8 中字符串分为单字节和双字节两种；
- (3) 间接字符串包括：ConsString、SlicedString、ThinString以及ExternalString。

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：www.zhihu.com/people/v8blink

本文由灰豆原创发布

转载出处：<https://www.anquanke.com/post/id/263879>

安全客 - 有思想的安全新媒体