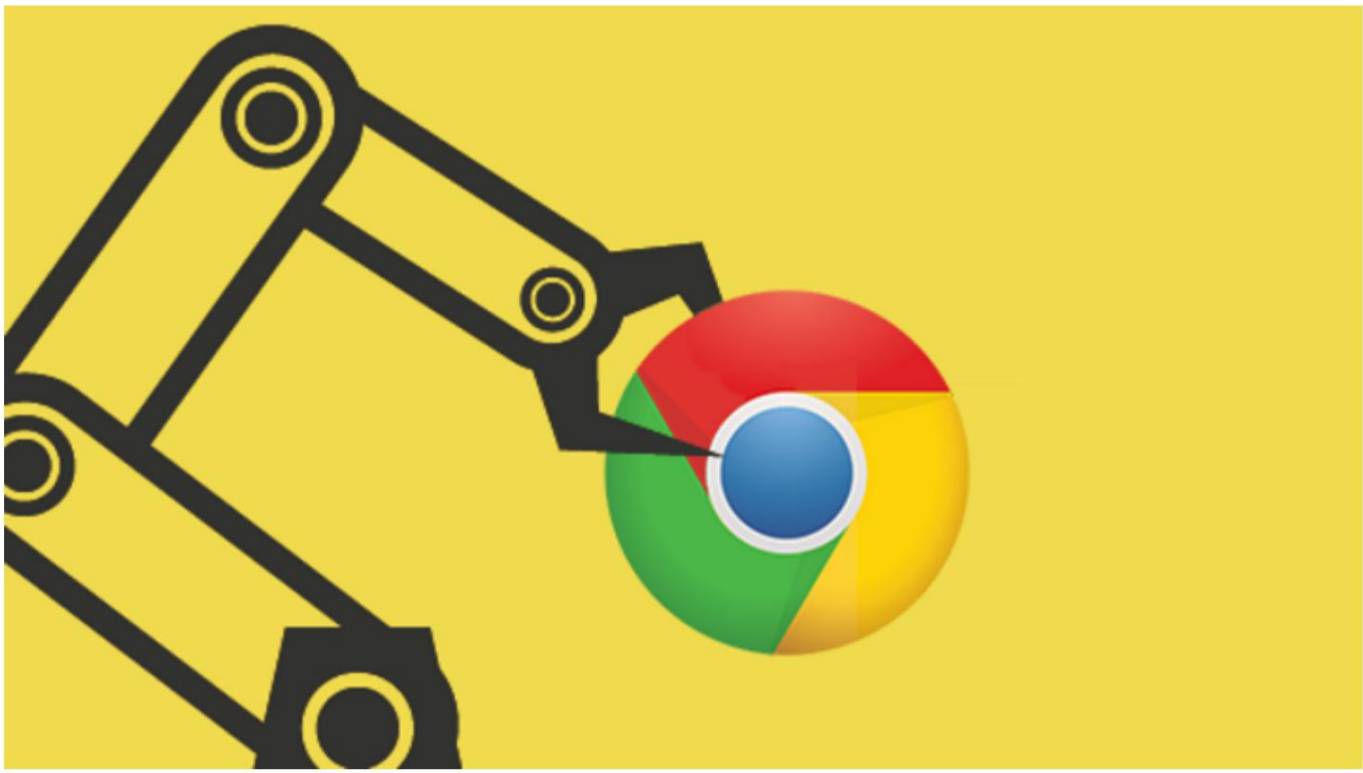
"Chrome V8 Source Code" 39. String.prototype.split source code analysis



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.split, and then demonstrates the calling, loading and execution process of

String.prototype.split 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.split source code

The test case code is as follows:

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

split() is implemented using TF_BUILTIN. The function name of replace() in V8 is StringPrototypeSplit, and the number is 596. The source code is as follows:

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

```
    const int kLimitArg = 1;
    TNode<IntPtrT> const argc =
        ChangeInt32ToIntPtr(Parameter(Descriptor::kJSActualArgumentsCount));
    CodeStubArguments args(this, argc);
    TNode<Object> receiver = args.GetReceiver();
    TNode<Object> const separator =
args.GetOptionalArgumentValue(kSeparatorArg);
    TNode<Object> const limit = args.GetOptionalArgumentValue(kLimitArg);
    TNode<NativeContext> context = CAST(Parameter(Descriptor::kContext));
    TNode<Smi> smi_zero = SmiConstant(0);
    RequireObjectCoercible(context, receiver, "String.prototype.split");
    MaybeCallFunctionAtSymbol(/*omitted....*/);
    TNode<String> subject_string = ToString_Inline(context, receiver);
    TNode<Number> limit_number = Select<Number>(
        IsUndefined(limit), [=] { return NumberConstant(kMaxUInt32); },
        [=] { return ToUint32(context, limit); });
    TNode<String> const separator_string = ToString_Inline(context, separator);
    Label return_empty_array(this);
    GotoIf(TaggedEqual(limit_number, smi_zero), &return_empty_array);
    {
      Label next(this);
      GotoIfNot(IsUndefined(separator), &next);
      const ElementsKind kind = PACKED_ELEMENTS;
      TNode<NativeContext> const native_context = LoadNativeContext(context);
      TNode<Map> array_map = LoadJSArrayElementsMap(kind, native_context);
      TNode<Smi> length = SmiConstant(1);
      TNode<IntPtrT> capacity = IntPtrConstant(1);
      TNode<JSArray> result = AllocateJSArray(kind, array_map, capacity,
length);
      TNode<FixedArray> fixed_array = CAST(LoadElements(result));
      StoreFixedArrayElement(fixed_array, 0, subject_string);
      args.PopAndReturn(result);
      BIND(&next);
    }
    {
      Label next(this);
      GotoIfNot(SmiEqual(LoadStringLengthAsSmi(separator_string), smi_zero),
                &next);
      TNode<Smi> subject_length = LoadStringLengthAsSmi(subject_string);
      GotoIf(SmiEqual(subject_length, smi_zero), &return_empty_array);
      args.PopAndReturn(
          StringToArray(context, subject_string, subject_length,
limit_number));
      BIND(&next);
    }
    TNode<Object> const result =
        CallRuntime(Runtime::kStringSplit, context, subject_string,
                    separator_string, limit_number);
    args.PopAndReturn(result);
    BIND(&return_empty_array);
    {
      const ElementsKind kind = PACKED_ELEMENTS;
      TNode<NativeContext> const native_context = LoadNativeContext(context);
      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. }
```

In the above code, the code receiver in line 7 represents the string str in the test case;

The 8th line of code separator represents "9" in the test case;

Line 9 of code reads limit;

Line 13 of the code: If separator is a regular expression, use MaybeCallFunctionAtSymbol() to complete the split operation;

Line 14 of the code converts receiver into a string and saves it to subject_string;

Line 15 of code converts the limit type to Number;

Line 16 of code converts separator into a string and saves it to separator_string;

The 20th line of code determines whether limit is equal to zero, returns an empty string if equal, and split exits;

Line 21 of code determines whether separator is equal to Undefined. If not, jumps to line 33 of code;

Lines 24-32 of code implement the ECMA regulations: "if {separator} is undefined, the result should be an array of size 1 containing the entire string. ";

The 37th line of code determines whether the length of the separator is equal to zero. If not, it jumps to the 43rd line of code;

Lines 39-41 of code are used to implement the split function when separator="" is used, that is, to create an array composed of all characters in the receiver;

Line 45 of code is used to implement the split function when the two conditions "separator length is greater than zero" and "limit is undefined or greater than zero" are met at the same time. able;

Lines 49-59 of code return an empty array.

The important functions used in StringPrototypeSplit are explained below:

(1) StringToArray method. In this article, this method implements the split function when separator="" is used. The source code is as follows:

```
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();
```

```
        BuildFastLoop(
            IntPtrConstant(0), length,
            [&](Node* index) {
              CSA_ASSERT(this, WordEqual(to_direct.PointerToData(&call_runtime),
                                         string_data));
              TNode<Int32T> char_code =
                  UncheckedCast<Int32T>(Load(MachineType::Uint8(), string_data,
                                             IntPtrAdd(index,
string_data_offset)));
              TNode<UintPtrT> code_index = ChangeUint32ToWord(char_code);
              TNode<Object> entry = LoadFixedArrayElement(cache, code_index);
              GotoIf(IsUndefined(entry), &fill_thehole_and_call_runtime);
              StoreFixedArrayElement(elements, index, entry);
            },
            1, ParameterMode::INTPTR_PARAMETERS, IndexAdvanceMode::kPost);
        TNode<Map> array_map = LoadJSArrayElementsMap(PACKED_ELEMENTS, context);
        result_array = AllocateJSArray(array_map, elements, length_smi);
        Goto(&done);
        BIND(&fill_thehole_and_call_runtime);
        {
          FillFixedArrayWithValue(PACKED_ELEMENTS, elements, IntPtrConstant(0),
                                  length, RootIndex::kTheHoleValue);
          Goto(&call_runtime); } }
      BIND(&call_runtime);
      {
        result_array = CAST(CallRuntime(Runtime::kStringToArray, context,
                                        subject_string, limit_number));
        Goto(&done);}
      BIND(&done);
      return result_array.value();}
```

In the above code, the 6th line of code determines whether subject_string is a single-byte string. If not, it jumps to line 43;

Line 8 of code calculates the array length. If limit is defined, the array length is the minimum of subject_string.length and limit; if not

Definition, the array length is equal to subject_string.length;

Line 14 of the code converts the indirect string into a direct string, and calls runtime processing when the conversion fails;

Line 15 of code allocates the array elements;

Lines 21-34 of the code use the BuildFastLoop method to fill elements, and use runtime processing when filling fails; note that BuildFastLoop uses

Turbofan optimization technology is used, please learn it by yourself.

Lines 40-45 of code use the runtime method to generate arrays;

(2) Runtime_StringSplit source code is as follows:

```
RUNTIME_FUNCTION(Runtime_StringSplit) {
int subject_length = subject->length();
int pattern_length = pattern->length();
CHECK_LT(0, pattern_length);
if (limit == 0xFFFFFFFFu) {//Omit...}
subject = String::Flatten(isolate, subject);
pattern = String::Flatten(isolate, pattern);
std::vector<int>* indices = GetRewoundRegexpIndicesList(isolate);
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 {
twenty one.         int part_start = 0;
twenty two.         FOR_WITH_HANDLE_SCOPE(isolate, int, i = 0, i, i < part_count, i++, {
twenty three.             int part_end = indices->at(i);
twenty four.             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) {//Omit...}
30. TruncateRegexpIndicesList(isolate);
31. return *result;
```

In the above code, the second line of code subject represents the test case string str;

The third line of code pattern represents separator;

Line 9 of code uses pattern to segment the subject and saves the segmentation results in indices. The result of segmentation is the number of slices and each slice

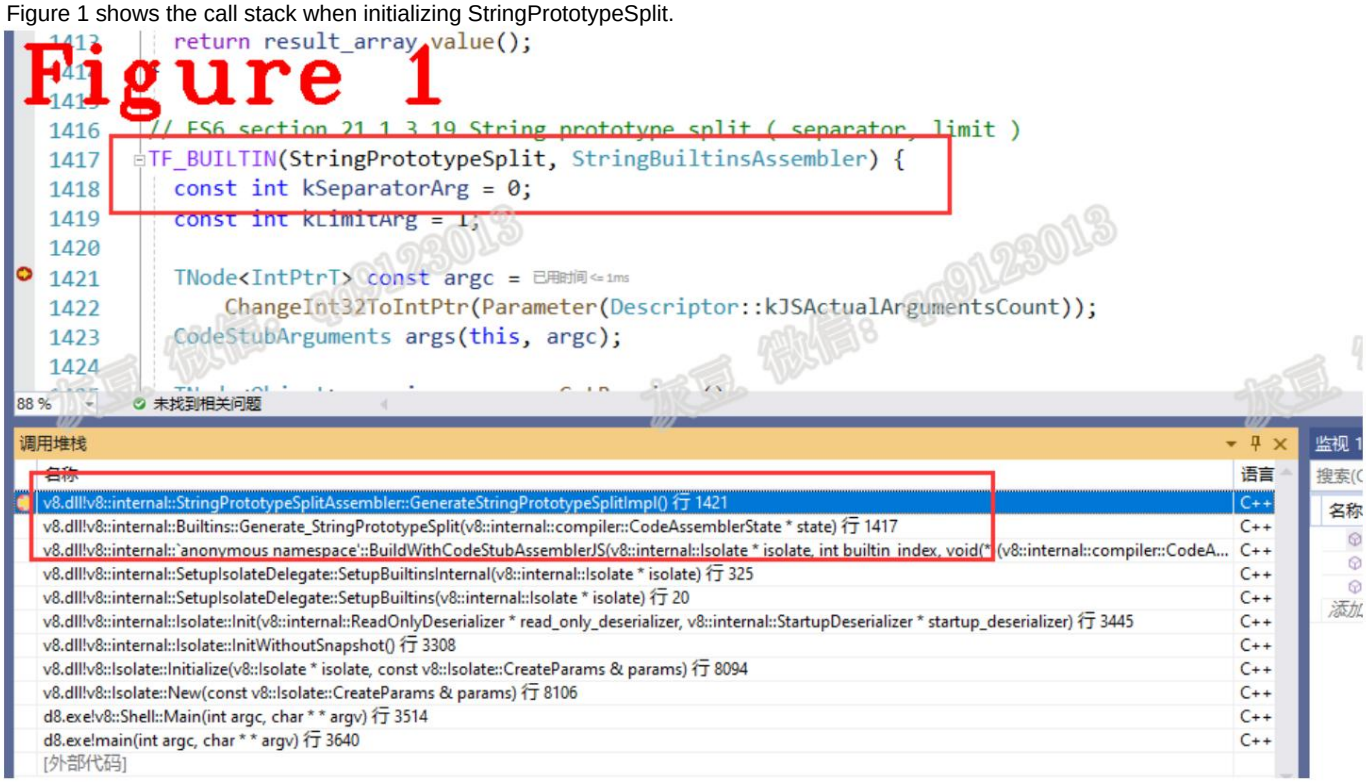
The starting and ending subscripts.

Line 13 of the code applies for the array elements, the length of which is equal to the number of slices;

When the number of code slices in line 19 is 1, copy the subject to elements;

Lines 20-28 use a loop to copy each slice into elements;

Figure 1 shows the call stack when initializing StringPrototypeSplit.



2 String.prototype.split test

The bytecode of the test code is as follows:

```
1. 000000A369D42A96 @ 16 : 12 01 2.          LdaConstant [1]
000000A369D42A98 @ 18 : 15 02 04 3.          StaGlobal [2], [4]
000000A369D42A9B @ 21 : 13 02 00 4.          LdaGlobal [2], [0]
000000A369D42A9E @ 24 : 26 f9 5.             Star r2
000000A369D42AA0 @ 26 : 29 f9 03 [3]         LdaNamedPropertyNoFeedback r2,

6. 000000A369D42AA3 @ 29 : 26 fa 7.          Star r1
000000A369D42AA5 @ 31 : 12 04 8.             LdaConstant [4]
000000A369D42AA7 @ 33 : 26 f8 9.             Star r3
000000A369D42AA9 @ 35 : 5f fa f9 02 10.      CallNoFeedback r1, r2-r3
000000A369D42AAD @ 39 : 15 05 06 11.           StaGlobal [5], [6]
000000A369D42AB0 @ 42 : 13 06 08 12.           LdaGlobal [6], [8]
000000A369D42AB3 @ 45 : 26 f9 13.              Star r2
000000A369D42AB5 @ 47 : 29 f9 07 [7]           LdaNamedPropertyNoFeedback r2,

14. 000000A369D42AB8 @ 50 : 26 fa 15.        Star r1
000000A369D42ABA @ 52 : 13 05 02 16.           LdaGlobal [5], [2]
000000A369D42ABD @ 55 : 26 f8 17.              Star r3
000000A369D42ABF @ 57 : 5f fa f9 02 18.      CallNoFeedback r1, r2-r3
000000A369D42AC3 @ 61 : 26 fb 19.            Star r0
000000A369D42AC5 @ 63 : ab 20. Constant      Return
pool (size = 8)
21. 000000A369D42A01: [FixedArray] in OldSpace
twenty two.      - map: 0x03abc3a00169 <Map>
twenty three.    - length: 8
```

```
24. 0: 0x00a369d429a1 <FixedArray[8]> 25. 1: 0x00a369d428c1
<String[#24]: How9are9you9doing9today?> 26. 2: 0x00a369d428a9 <String[#3]: str> 27. 3: 0x023319
5eba31 <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>
```

In the above code, lines 1-4 read the string "How9are9you9doing9today?" and save it to the r2 register; lines 5-6 get the string method split and save it to the r1 register; lines 7-8 get separator
string and save it to the r3 register. The separator of the test case is '9'. Line 9 of code CallNoFeedback calls the split method and passes two parameters r2 and r3 to the split method.

Technical

summary (1) In most cases, the split method is implemented by runtime; **(2)** Strings in v8 are divided into single-byte and double-byte types; **(3)** Indirect strings include: ConsString, SlicedString, ThinString and ExternalString.

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

Personal abilities are limited and there are shortcomings and

mistakes. Welcome to criticize and correct me. WeChat: qq9123013 Note: v8 Communication Zhihu:

www.zhihu.com/people/

v8blink This article was originally published by Gray Bean and reprinted from: https://

www.anquanke.com/ post/id/263879 Safe Guest - Thoughtful new security media