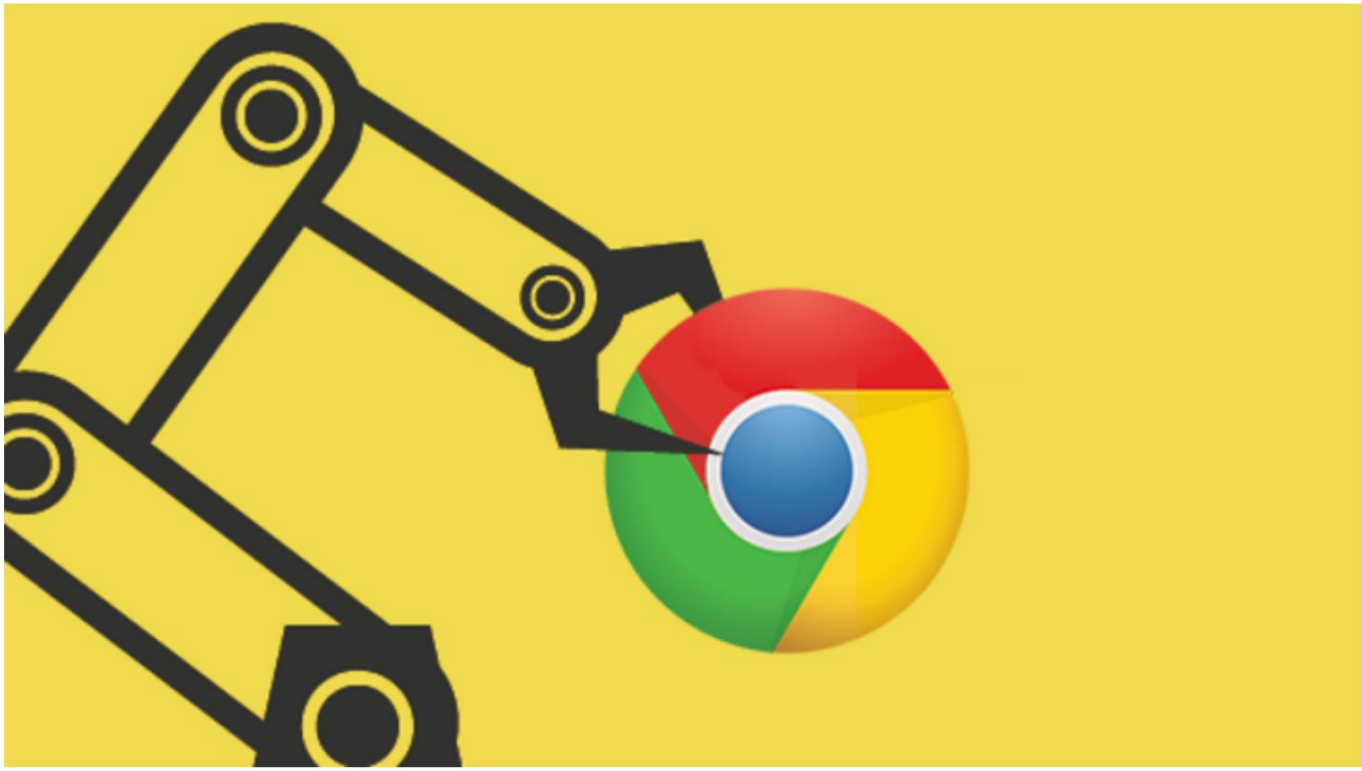
"Chrome V8 Source Code" 37. String.prototype.match source code analysis



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

String.prototype.match 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.match source code

The test case code is as follows:

```
var str="1 plus 2 equal 3"; str.match(/
\d+/g);
```

match() is implemented using TF_BUILTIN. The function name of concet() in V8 is StringPrototypeMatch, and the number is 591. The source code is as follows:

```
1. TF_BUILTIN(StringPrototypeMatch, StringMatchSearchAssembler) { 2. TNode<Object>
receiver = CAST(Parameter(Descriptor::kReceiver));
```

```
3. TNode<Object> maybe_regexp = CAST(Parameter(Descriptor::kRegexp));
4. TNode<Context> context = CAST(Parameter(Descriptor::kContext));
5. Generate(kMatch, "String.prototype.match", receiver, maybe_regexp, context);}
6. //Separation.....
7. void Generate(Variant variant, const char* method_name,TNode<Object> receiver,
TNode<Object> maybe_regexp, TNode<Context> context) {
8.        Label call_regexp_match_search(this);
9.         Builtins::Name builtin;
10.           Handle<Symbol> symbol;
11.           DescriptorIndexNameValue property_to_check;
12.           if (variant == kMatch) {
13.              builtin = Builtins::kRegExpMatchFast;
14.              symbol = isolate()->factory()->match_symbol();
15.              property_to_check = DescriptorIndexNameValue{
16.                    JSRegExp::kSymbolMatchFunctionDescriptorIndex,
17.                    RootIndex::kmatch_symbol, Context::REGEXP_MATCH_FUNCTION_INDEX};
18.           } else {//Omit.....
19.          }
              RequireObjectCoercible(context, receiver, method_name);//province
20.Slightly .....
twenty one.           { RegExpBuiltinsAssembler regexp_asm(state());
twenty two.              TNode<String> receiver_string = ToString_Inline(context, receiver);
twenty three.            TNode<NativeContext> native_context = LoadNativeContext(context);
twenty four.             TNode<HeapObject> regexp_function = CAST(
25.                   LoadContextElement(native_context,
Context::REGEXP_FUNCTION_INDEX));
26.              TNode<Map> initial_map = CAST(LoadObjectField(
27.                   regexp_function, JSFunction::kPrototypeOrInitialMapOffset));
28.              TNode<Object> regexp = regexp_asm.RegExpCreate(
29.                   context, initial_map, maybe_regexp, EmptyStringConstant());
30.              Label fast_path(this), slow_path(this);
31.              regexp_asm.BranchIfFastRegExp(context, CAST(regexp), initial_map,
32.                   PrototypeCheckAssembler::kCheckPrototypePropertyConstness,
33.                   property_to_check, &fast_path, &slow_path);
34.              BIND(&fast_path);
35.              Return(CallBuiltin(builtin, context, regexp, receiver_string));
36.              BIND(&slow_path);
37.              {
38.                 TNode<Object> maybe_func = GetProperty(context, regexp, symbol);
39.                 Callable call_callable = CodeFactory::Call(isolate());
40.                 Return(CallJS(call_callable, context, maybe_func, regexp,
41.                                    receiver_string));
42.              } } }
```

In the above code, lines 1-5 are the entry functions of match(); Generate() (line 7 of code) is used to implement the match function, and the parameter variant The value of can only be Match or Search, which shows that Search is also implemented by Generate(). The parameter receiver is a string (in the test case str), maybe_regexp is a regular string (/\d+/g in the test case);

Lines 13-17 of the code prepare three parameters: Builtins::kRegExpMatchFast, symbol and property_to_check, among which kRegExpMatchFast and symbol will be used in fast regularization;

Line 22 of the code converts receiver into a string and stores it in receiver_string;

Lines 23-28 use the string (/\d+/g) to create the regular expression regexp;

The 31st line of code determines whether the conditions for using fast regular matching are met. If it is met, the 35th line of code is executed. Otherwise, the 36-40 lines of code are executed.

code;

Line 35 of code performs fast regular matching; **Tips:** The regularity implemented using Builtin is called fast regular matching;

Lines 36-40 of code perform slow regular matching.

The important functions in Generate() are explained below:

(1) Builtins::kRegExpMatchFast is used to implement fast regular matching. The source code is as follows:

```
1. TF_BUILTIN(RegExpMatchFast, CodeStubAssembler) {
2.         compiler::CodeAssemblerState* state_ = state(); compiler::CodeAssembler
ca_(state());
3.         TNode<Context> parameter0 = UncheckedCast<Context>
(Parameter(Descriptor::kContext));
4.         USE(parameter0);
5. compiler::TNode<JSRegExp> parameter1 = UncheckedCast<JSRegExp>
(Parameter(Descriptor::kReceiver));
6.         USE(parameter1);
7.         compiler::TNode<String> parameter2 = UncheckedCast<String>
(Parameter(Descriptor::kString));
8.         USE(parameter2);
9.         compiler::CodeAssemblerParameterizedLabel<Context, JSRegExp, String>
block0(&ca_, compiler::CodeAssemblerLabel::kNonDeferred);
10.             ca_.Goto(&block0, parameter0, parameter1, parameter2);
11.         if (block0.is_used()) {
12.             compiler::TNode<Context> tmp0;
13.             compiler::TNode<JSRegExp> tmp1;
14.             compiler::TNode<String> tmp2;
15.             ca_.Bind(&block0, &tmp0, &tmp1, &tmp2);
16.             ca_.SetSourcePosition("../../../src/builtins/regexp-match.tq", 27);
17.             compiler::TNode<Object> tmp3;
18.             USE(tmp3);
19.             tmp3 = FastRegExpPrototypeMatchBody_322(state_, compiler::TNode<Context>
{tmp0}, compiler::TNode<JSRegExp>{tmp1}, compiler::TNode<String>{tmp2});
20.             CodeStubAssembler(state_).Return(tmp3);
twenty one.         }
twenty two. }
```

In the above code, lines 3-8 define three parameters: context (parameter0), regular (parameter1) and string (parameter2).

number;

Lines 10-14 of code Goto are used to jump to block0. Among them, tmp1 represents parameter1 and tmp2 represents parameter2;

Line 19 of code FastRegExpPrototypeMatchBody_322() is the entry function, which is called in this function

RegExpBuiltinsAssembler::RegExpPrototypeMatchBody completes regular matching, which will be explained separately in subsequent articles.

(2) BranchIfFastRegExp determines whether the fast regular conditions are met. The source code is as follows:

```
1. void RegExpBuiltinsAssembler::BranchIfFastRegExp(/*Omit...*/) {
2.         CSA_ASSERT(this, TaggedEqual(LoadMap(object), map));
3.         GotoIfForceSlowPath(if_ismodified);
4.         TNode<NativeContext> native_context = LoadNativeContext(context);
5.         GotoIf(IsRegExpSpeciesProtectorCellInvalid(native_context), if_ismodified);
6.         TNode<JSFunction> regexp_fun =
```

```
7.              CAST(LoadContextElement(native_context,
Context::REGEXP_FUNCTION_INDEX));
8.      TNode<Map> initial_map = CAST(
9.              LoadObjectField(regexp_fun, JSFunction::kPrototypeOrInitialMapOffset));
10.     TNode<BoolT> has_initialmap = TaggedEqual(map, initial_map);
11.     GotoIfNot(has_initialmap, if_ismodified);
12.     TNode<Object> last_index = FastLoadLastIndexBeforeSmiCheck(CAST(object));
13.     GotoIfNot(TaggedIsPositiveSmi(last_index), if_ismodified);
14.     // Verify the prototype.
15.     TNode<Map> initial_proto_initial_map = CAST(
16.             LoadContextElement(native_context,
Context::REGEXP_PROTOTYPE_MAP_INDEX));
17.     DescriptorIndexNameValue properties_to_check[2];
18.     int property_count = 0;
19.     properties_to_check[property_count++] = DescriptorIndexNameValue{
20.             JSRegExp::kExecFunctionDescriptorIndex, RootIndex::kexec_string,
twenty one.             Context::REGEXP_EXEC_FUNCTION_INDEX};
twenty two.     if (additional_property_to_check) {
twenty three.       properties_to_check[property_count++] = *additional_property_to_check;
twenty four.    }
25.     PrototypeCheckAssembler prototype_check_assembler(
26.             state(), prototype_check_flags, native_context,
initial_proto_initial_map,
27.             Vector<DescriptorIndexNameValue>(properties_to_check, property_count));
28.     TNode<HeapObject> prototype = LoadMapPrototype(map);
29.     prototype_check_assembler.CheckAndBranch(prototype, if_isunmodified,
30.                                              if_ismodified);
31. }
```

In the above code, if_ismodified represents slow regularity; the 3rd line of code GotoIfForceSlowPath is based on

V8_ENABLE_FORCE_SLOW_PATH determines whether to use slow regular expression;

The second line of code detects the tag tag of the regular expression object map;

Lines 10-11 of the code determine whether the tag of the regular expression object is equal to the tag of regexp_fun in native_context;

Lines 15-29 of code detect the prototype attribute and decide whether to use fast regularization based on the detection results.

(3) The source code of the MaybeCallFunctionAtSymbol method is as follows:

```
1. void StringBuiltinsAssembler::MaybeCallFunctionAtSymbol(
2.         Node* const context, Node* const object, Node* const maybe_string,
3.         Handle<Symbol> symbol,
4.         DescriptorIndexNameValue additional_property_to_check,
5.         const NodeFunction0& regexp_call, const NodeFunction1& generic_call) {
6.       Label out(this);
7.       // Smis definitely don't have an attached symbol.
8.       GotoIf(TaggedIsSmi(object), &out);
9.       {
10.          Label stub_call(this), slow_lookup(this);
11.          GotoIf(TaggedIsSmi(maybe_string), &slow_lookup);
12.          GotoIfNot(IsString(maybe_string), &slow_lookup);
13.          RegExpBuiltinsAssembler regexp_asm(state());
14.          regexp_asm.BranchIfFastRegExp(
15.                  CAST(context), CAST(object), LoadMap(object),
```

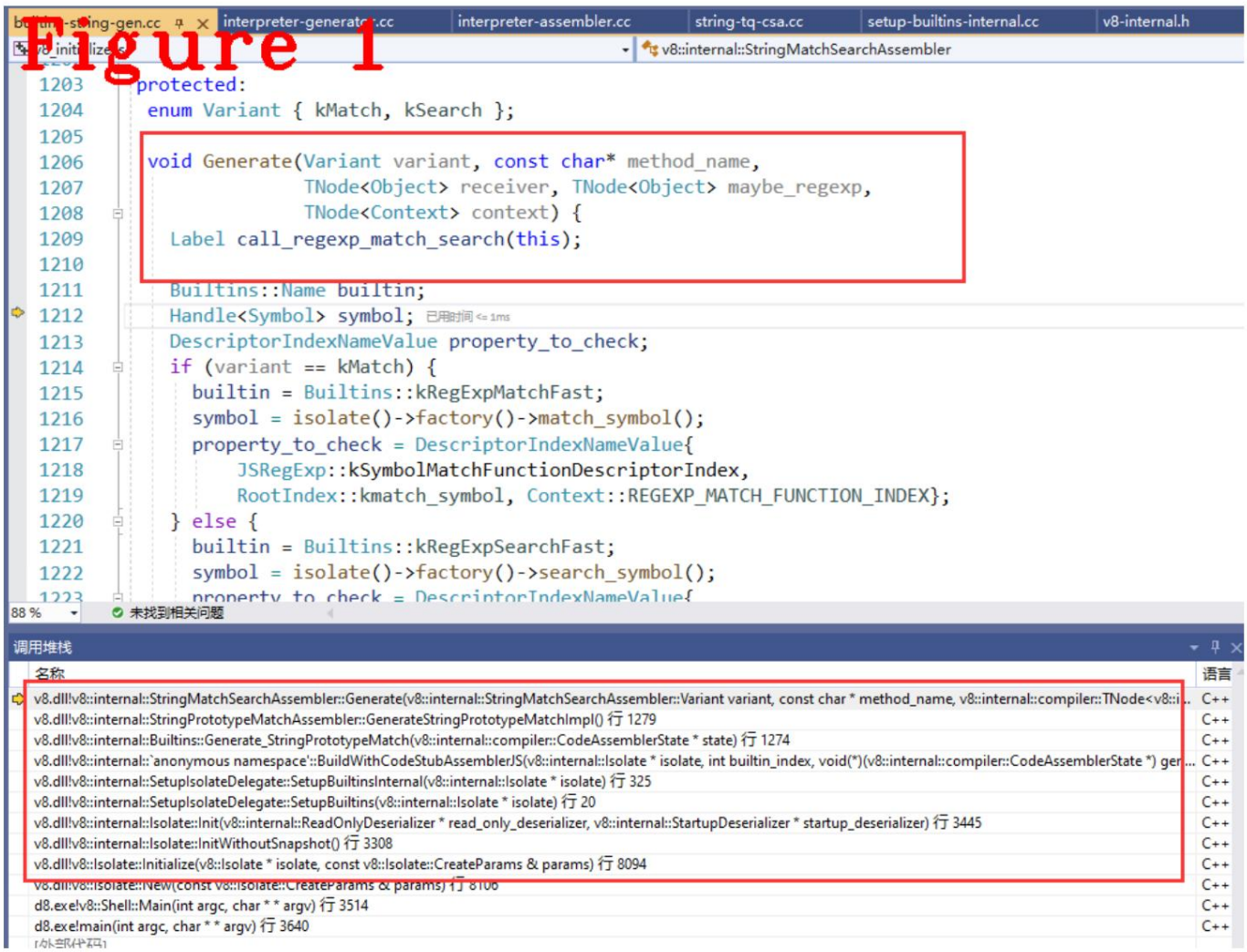
```
16.             PrototypeCheckAssembler::kCheckPrototypePropertyConstness,
17.             additional_property_to_check, &stub_call, &slow_lookup);
18.         BIND(&stub_call);
19. .
20.         regexp_call();
twenty one.     BIND(&slow_lookup);
twenty two.   }
twenty three. GotoIf(IsNullOrUndefined(object), &out);
twenty four.  TNode<Object> const maybe_func = GetProperty(context, object, symbol);
25.           GotoIf(IsUndefined(maybe_func), &out);
26.           GotoIf(IsNull(maybe_func), &out);
27.           // Attempt to call the function.
28.           generic_call(maybe_func);
29.           BIND(&out);
30. }
```

Lines 11-12 in the above code determine whether the regular expression is SMI or String. If the result is true, the slow regular expression is executed;

Line 14 of the code BranchIfFastRegExp determines whether the prototype chain attributes meet the fast regularity condition;

Lines 23, 25, and 26 of code respectively determine whether the string is empty and whether the regular expression is undefined or empty.

Figure 1 shows the function call stack of Generate.



3 String.prototype.match test

The bytecode of the test case is as follows:

```
1. //Omit.....
2. 0000038004E42A8E @ 16 : 12 01 3.          LdaConstant [1]
0000038004E42A90 @ 18 : 15 02 04 4.          StaGlobal [2], [4]
0000038004E42A93 @ 21 : 13 02 00 5.          LdaGlobal [2], [0]
0000038004E42A96 @ 24 : 26 f9 6.             Star r2
0000038004E42A98 @ 26 : 29 f9 03 7.          LdaNamedPropertyNoFeedback r2, [3]
0000038004E42A9B @ 29 : 26 fa 8.             Star r1
0000038004E42A9D @ 31 : 79 04 06 01 9.       CreateRegExpLiteral [4], [6], #1
0000038004E42AA1 @ 35 : 26 f8 10.            Star r3
0000038004E42AA3 @ 37 : 5f fa f9 02 11.      CallNoFeedback r1, r2-r3
0000038004E42AA7 @ 4 1 : 15 05 07 12.        StaGlobal [5], [7]
0000038004E42AAA @ 44 : 13 06 09 13.         LdaGlobal [6], [9]
0000038004E42AAD @ 47 : 26 f9 14.            Star r2
0000038004E42AAF @ 49 : 29 f9 07 [7]         LdaNamedPropertyNoFeedback r2,

15. 0000038004E42AB2 @ 52 : 26 fa 16.        Star r1
0000038004E42AB4 @ 54 : 13 05 02 17.         LdaGlobal [5], [2]
0000038004E42AB7 @ 57 : 26 f8 18.            Star r3
0000038004E42AB9 @ 59 : 5f fa f9 02 19.      CallNoFeedback r1, r2-r3
0000038004E42ABD @ 63 : 26 fb 20.            Star r0
0000038004E42ABF @ 65 : ab 21. Constant      Return
pool (size = 8)
22. 0000038004E429F9: [FixedArray] in OldSpace
twenty three.       - map: 0x01afd2dc0169 <Map>
twenty four.        - length: 8
25. 0: 0x038004e42999 <FixedArray[8]>
26. 1: 0x038004e428c1 <String[#16]: 1 plus 2 equal 3>
27. 2: 0x038004e428a9 <String[#3]: str>
28. 3: 0x022bdecab4b9 <String[#5]: match>
29. 4: 0x038004e428f9 <String[#3]: \d+>
30. 5: 0x038004e428e1 <String[#3]: res>
31. 6: 0x022bdecb3699 <String[#7]: console>
32. 7: 0x022bdecb2cd9 <String[#3]: log>
33. Handler Table (size = 0)
```

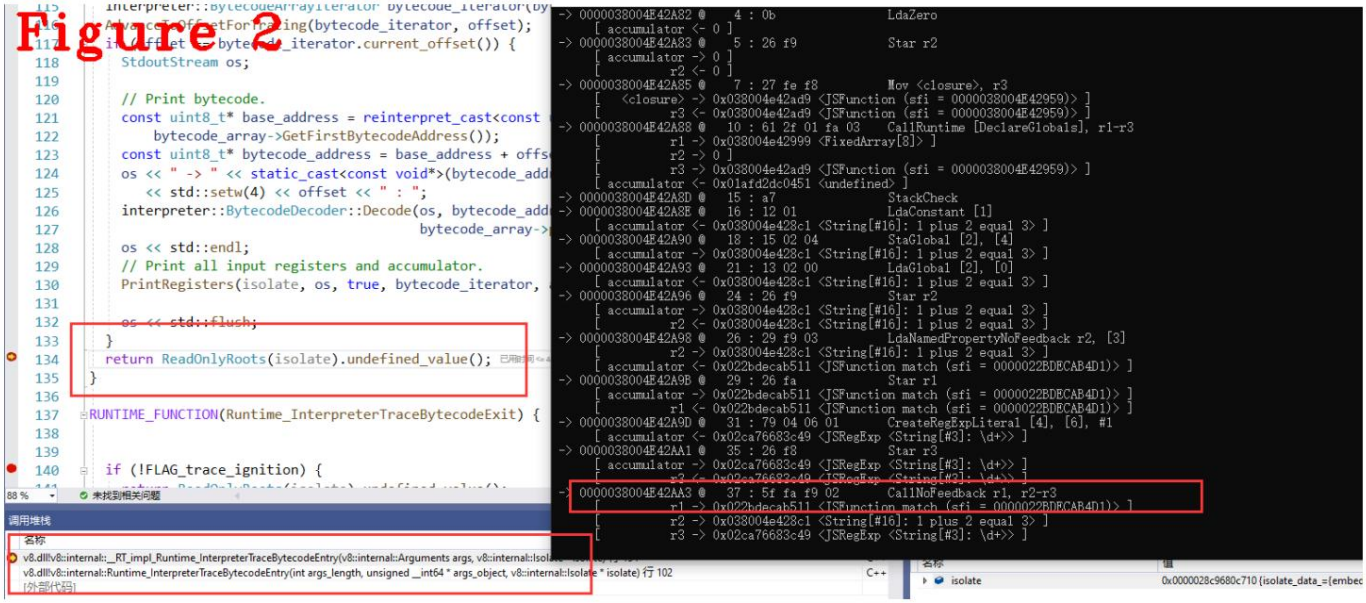
In the above code, lines 2-5 load "1 plus 2 equal 3" to the r2 register;

Line 6 of code obtains the string method match and stores it in the r1 register;

The 8th line of code creates a positive expression object for the string \d+ and stores it in the r3 register;

Line 10 of the code CallNoFeedback calls the match method (r1 register) and passes the two parameters r2 and r3 to the match method.

Figure 2 shows the entry of the bytecode CallNoFeedback. From here on, you can see the matching process of the regular expression.



Technical

summary (1) Fast regex is a fast match implemented using Builtins::kRegExpMatchFast; **(2)** The

judgment conditions for using fast regex include: whether the string type is correct, the type of regular expression,

V8_ENABLE_FORCE_SLOW_PATH etc.

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 Exchange Zhihu: www.zhihu.com/people/v8blink

This article was originally

published by Huidou and reprinted from: https://www.anquanke.com/post/id/263786

Anquanke - Thoughtful new security media