QQ邮箱 mail.qq.com

[quickjs-devel] Re: A possible issue about the scope of variablein QuickJS

发件人：Fabrice Bellard<fabrice@bellard.org>
时　间：2019年9月17日(星期二) 凌晨2:28
收件人：QuickJS<quickjs-devel@freelists.org>

Hi,

It is a bug in QuickJS and it will be fixed in the next release. Thank
you for the report !

Best regards,

Fabrice.

On 9/16/19 3:07 AM, Houyou Yao wrote:
> Hi!
> Perhaps the reason for this issue is that QuickJS does not meet the
> ES standard，but I have some questions listed below:
> Q1: Is it a bug of QuickJS?
> Q2: if Q1 is yes, will this bug be fixed in the future, or has it been
> fixed?
> Looking forward to your replay, thank you very much.
>
> Cheers,
> Houyou Yao
>
>
> ------------------ Original ------------------
> *From: * "Mario Gliewe";<mag@nohup.org>;
> *Send time:* Saturday, Sep 7, 2019 4:12 AM
> *To:* "quickjs-devel"<quickjs-devel@freelists.org>;
> *Subject: * [quickjs-devel] Re: A possible issue about the scope of
> variablein QuickJS
>
>> Looks like what's missing is 6.a.: SameValue(func, %eval%) == true during
>> execution of bytecode.
>>
>> https://tc39.es/ecma262/#sec-function-calls-runtime-semantics-evaluation
>>
>> regards,
>> o.
>

> thx for clearifying, i guess thats exactly it. so the actual testcase
> would be:
>
> --snip--
>
> let x_eval = eval;
> eval = () => {};
> x_eval('console.log("ok")');
> eval('console.log("oops")');
>
> --snap--
>
> which works in node but fails on qjs.
>
> greets
>
> maG
>
>
> On 06.09.19 18:17, Ond?ej Jirman wrote:
>> On Fri, Sep 06, 2019 at 05:59:39PM +0200, quickjs mailing list wrote:
>>> Hi,
>>>
>>> On Fri, Sep 06, 2019 at 04:22:40PM +0200, Mario Gliewe wrote:
>>>> very funny, seems to be specific to the eval() function..
>>>>
>>>> i tried the following, which passed ok:
>>>>
>>>> --snip--
>>>>
>>>> let evil=666;
>>>> function test1(evil) {
>>>> console.log('the evil is ',evil);
>>>> eval('console.log("the evil is ",evil)');
>>>> }
>>>> function test2() {
>>>> test1(42);
>>>> };
>>>> test2();
>>>>
>>>>
>>>> function test3(Object) {
>>>> console.log('Object is ', Object);
>>>> }
>>>> function test4() {

```
>>>>        console.log('Object is ', Object);
>>>> }
>>>> function test5() {
>>>>        test3(' void');
>>>> };
>>>> test4();
>>>> test5();
>>>>
>>>> --snap--
>>> It's due to special handling in js_parse_postfix_expr. If it's named
> eval,
>>> it will do eval, no matter what. And this is decided at parse time
> and not
>>> during execution:
>> Looks like what's missing is 6.a.: SameValue(func, %eval%) == true during
>> execution of bytecode.
>>
>>   https://tc39.es/ecma262/#sec-function-calls-runtime-semantics-evaluation
>>
>> regards,
>> o.
>>
>>>           if (call_type == FUNC_CALL_NORMAL) {
>>>           parse_func_call:
>>>              switch(opcode = get_prev_opcode(fd)) {
>>>              case OP_get_field:
>>>                 /* keep the object on the stack */
>>>                 fd->byte_code.buf[fd->last_opcode_pos] =
> OP_get_field2;
>>>                 break;
>>>              case OP_scope_get_private_field:
>>>                 /* keep the object on the stack */
>>>                 fd->byte_code.buf[fd->last_opcode_pos] =
> OP_scope_get_private_field2;
>>>                 break;
>>>              case OP_get_array_el:
>>>                 /* keep the object on the stack */
>>>                 fd->byte_code.buf[fd->last_opcode_pos] =
> OP_get_array_el2;
>>>                 break;
>>>              case OP_scope_get_var:
>>>                 {
>>>                    JSAtom name;
>>>                    int scope;
>>>                    name = get_u32(fd->byte_code.buf +
```

```
> fd->last_opcode_pos + 1);
>>>                     scope = get_u16(fd->byte_code.buf +
> fd->last_opcode_pos + 5);
>>> here >>>               if (name == JS_ATOM_eval && call_type ==
> FUNC_CALL_NORMAL) {
>>>                         /* direct 'eval' */
>>>                         JS_FreeAtom(s->ctx, name);
>>>                         fd->byte_code.size = fd->last_opcode_pos;
>>>                         fd->last_opcode_pos = -1;
>>>                         opcode = OP_eval;
>>>                     } else {
>>>                         /* verify if function name resolves to a
> simple
>>>                            get_loc/get_arg: a function call
> inside a `with`
>>>                            statement can resolve to a method call
> of the
>>>                            `with` context object
>>>
>>> regards,
>>> o.
>>>
>>>> greets
>>>>
>>>> maG
>>>>
>>>> On 06.09.19 14:35, 姚厚友 wrote:
>>>>> Dear Sir or Madam:
>>>>>   I have a doubt about QuickJS, the detailed description is as follows:
>>>>>
>>>>>
>>>>> ## version: quickjs-2019-08-18
>>>>>
>>>>> ## Testcase:
>>>>> var NISLFuzzingFunc = function(eval) {
>>>>>     print(eval("print(false);"));
>>>>> };
>>>>> var NISLParameter1 = function() {
>>>>>     print("run to NISLParameter1");
>>>>>     return true;
>>>>> };
>>>>> NISLFuzzingFunc(NISLParameter1);
>>>>>
>>>>> ## Command:
>>>>> ./ quickjs-2019-08-18/qjs testcase.js
```

>>>>>
>>>>> ## Output:
>>>>> false
>>>>> undefined
>>>>>
>>>>> ## Expected output:
>>>>> run to NISLParameter1
>>>>> true
>>>>>
>>>>> ## Description:
>>>>> In the testcase above, "eval" (Line 2) should be the formal
>>>>> parameter "eval", but QuickJS treats it as a global object "eval".
>>>>> According to the scope of variable, I think this may be a bug in
> QuickJS.
>>>>> Do you think so? Looking forward to your reply.
>>>>>
>>>>> your faithfully
>>>>> houyou yao
>>>>>
>>>>>
>>>>>
>