

ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

CENTER OF INFORMATION TECHNOLOGY AND SCIENTIFIC

COMPUTING

Assignment I

Name: Mikiyas Daniel Tefera

Id: ATR/1876/11

Section:It

To: Fitum Almu

January 2020

Contents

[IS JAVA INTERPRETED LANGUAGE IN IT ENTIRETY? 4](#_Toc62314273)

[How does javascript work? 4](#_Toc62314274)

[Just-in-time compiler (JIT) 4](#_Toc62314275)

[Conclusion 5](#_Toc62314276)

# IS JAVA INTERPRETED LANGUAGE IN IT ENTIRETY?

To answer this question we must look at how javascript works in the first place.

## How does javascript work?

For starters, javascript is supposed to run on browsers so for the browser to understand the language it has a javascript engine.

Javascript engine is a program that goes through the javascript code, character by character, and transforms it to a language that the computer CPU understands and executes which is machine code.

The javascript engine executes the javascript line by line which makes it go through the whole code blindly consequently making it redundant and inefficient, which means the javascript engine has to keep retranslating the same code like when you are in a loop which makes the execution slow. This where JIT comes.

### Just-in-time compiler (JIT)

As a way of getting rid of javascript interpreter inefficiency browsers started mixing compilers in.

Different browsers use different ways but the idea is still the same. They added a new part to the engine, called a monitor or also known as a profiler.

The profiler is used to watch the code as it runs making a note of how many times it runs and types used, so at first, the monitor just runs everything through the interpreter.

If the same lines of code are run multiple times the segment of codes is called hot.

When the part of the code is getting the getting very hot the monitor will send it off to the optimizing compiler which will make it complied and stored consequently making a faster version of the code, to do that the compiler has to make some assumptions.

For example, if it can assume that all objects created by a particular constructor have the same shape-that is, that they always have the same property names and that those properties were added in the same order – then it can continue to be true.

But this might not always be true a code can have 99 objects that all have the same shapes, but the 100th might be missing a property.

So the compiled code needs to check before it runs to see whether the assumption is valid. if they are, then the compiled code runs .but if not, the jit assumes that it made the wrong assumptions and trashes the optimized code. which makes the execution go back to the interpreter. this process is called deoptimization.

Even if JIT is supposed to make the code run faster it can cause unexpected performance problems, to avoid that browsers have limits to break out of this optimization. deoptimzation cycle when needed.

## Conclusion

So, in my opinion, even if JIT acts as an optimizer for the javascript code we can't deny the fact that the java engine uses basic definitions of compiled language along the way, which paves the way for usage of both compiled and interpreted, so to address this question javascript isn’t interpreted language in its entirety but **depends on its implantation,** which means it depends on how the java engine choose to implement it, it could be solely interpreted, compiled or both at the same time, which most java engines do nowadays.

# The history of “typeof null”

In javascript, typeof null returns an object, which suggests that its an object when it is a primitive value.

In the first version of "typeof null," the bug comes from the JavaScript error achieve. In this version, with a 32-bit value is stored, the actual data comprising a type of marker (1-3) and a representation of the value. On the lower, a total of five kinds of tag storage type:

* 000: **object**. The data is a reference to an object.
* 1:**init**.the data is a 31 bit signed integer.
* 010:**double**. The data is a reference to a double floating-point number.
* 100:**string**.the data is a reference to a string.
* 110:**Boolean**. The data is a Boolean.

If the lower bit is 1, then only one type of flag bit long; if 0, then labeled with a 3- bit type, two additional bit provide four types

Two special values:

* To define **undefined**  (JSVAL\_VOID) integer -230(An integer number outside the range).
* To define **null**  (JSVAL\_NULL) the machine code is a null pointer. or: an object type tag plus a reference that zero.

**Code of tyeof**

**JS\_PUBLIC\_API(JSType)**

**JS\_TypeOfValue(JSContext \*cx, jsval v)**

**{**

**JSType type = JSTYPE\_VOID;**

**JSObject \*obj;**

**JSObjectOps \*ops;**

**JSClass \*clasp;**

**CHECK\_REQUEST(cx);**

**if (JSVAL\_IS\_VOID(v)) { // (1)**

**type = JSTYPE\_VOID;**

**} else if (JSVAL\_IS\_OBJECT(v)) { // (2)**

**obj = JSVAL\_TO\_OBJECT(v);**

**if (obj &&**

**(ops = obj->map->ops,**

**ops == &js\_ObjectOps**

**? (clasp = OBJ\_GET\_CLASS(cx, obj),**

**clasp->call || clasp == &js\_FunctionClass) // (3,4)**

**: ops->call != 0)) { // (3)**

**type = JSTYPE\_FUNCTION;**

**} else {**

**type = JSTYPE\_OBJECT;**

**}**

**} else if (JSVAL\_IS\_NUMBER(v)) {**

**type = JSTYPE\_NUMBER;**

**} else if (JSVAL\_IS\_STRING(v)) {**

**type = JSTYPE\_STRING;**

**} else if (JSVAL\_IS\_BOOLEAN(v))**

**type = JSTYPE\_BOOLEAN;**

**}**

**return type;**

**}**

The code executes like this :

* In (1), the engine is first checked whether the undefined value V(VOID), By checking whether the comparison value is equal to;

#define JSVAL\_IS\_VOID(v) ((v) == JSVAL\_VOID)

* Next check (2) if there is a value of the object tag. If it can be called. (3) or its internal property [[class]] mark it as a function of (4), then v is a function. Otherwise, it is an object. This is the result of typeof null generated.
* The next check number, string, and Boolean values. Not even explicitly check if it is null.

#define JSVAL\_IS\_NULL(v) ((v) == JSVAL\_NULL)

This bug isn’t removed yet and even for the foreseeable future as it will break the existing code that relies exactly on this principle, which means that every web application out there will need to undergo a refactoring.