

Yes, AFAICT, everybody in the room yesterday agreed that option 1 would be preferable in almost every way.



bterIson commented on Jan 29



@rossberg-chromium true, iff:

- 1. we can't remove [[Enumerate]] trap, and
- 2. we are ok standardizing that keys reconfigured to be non-enumerable during for-in enumeration are still visited (both for proxies and for regular objects).

I'm not sure I'm ok with 2, yet. And I'm not sure we had consensus on this. Am I remembering wrong?



erights commented on Jan 29



What I think I remember regarding @bterlson 's question 2: I remember that when we tried to declare consensus, you (@bterlson) said you had to check first what the implementations issues were with Chakra. But if there was no cause for concern there, I think you provisionally agreed that this would be ok.

I do not remember anyone else expressing reservations. But we did not try to officially declare conditional consensus (as we did with the enumerate trap itself), so not everyone with reservations may have felt the need to voice them.

Does this sound right?



bterison commented on Jan 29



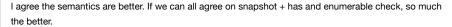


That sounds right @erights. Also I have discovered (see above) that Chakra semantics with re-configured non-enumerable keys are handled same as delete (ie. will not be visted). IMO this seems like better semantics to me. I will need to investigate more to see if this is something we could get away with changing.



erights commented on Jan 29





But I am much more interested in all of us agreeing on the same behavior than I am at improving this kind of semantic issue. +1 to whatever more deterministic behavior we can all agree on. Ideal would be to agree to a deterministic behavior.



allenwb commented on Jan 29

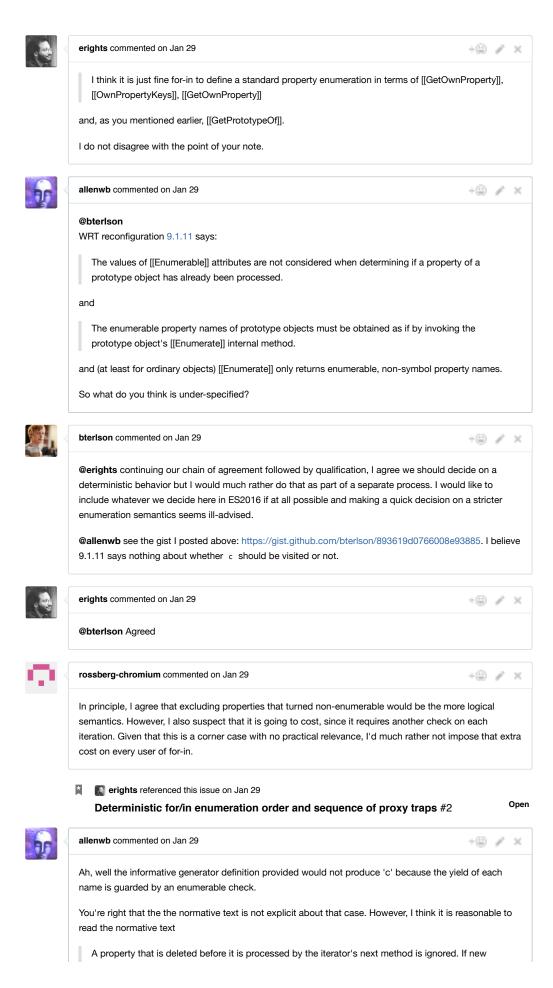


I think the place to virtualize the prototype chain (or to implement unique inheritance semantics) is in [[GetPrototypeOf]], [[OwnPropertyKeys], [[GetOwnProperty]], etc. Having to have [[Enumerate]] separately implement it is a bug farm. For example, the part of [ordinary object [Enumerate] that says

The enumerable property names of prototype objects must be obtained as if by invoking the prototype object's [[Enumerate]] internal method. [[Enumerate]] must obtain the own property keys of the target object as if by calling its [[OwnPropertyKeys]] internal method. Property attributes of the target object must be obtained as if by calling its [[GetOwnProperty]] internal method.

was added very late in response to a bug report.

I think it is just fine for-in to define a standard property enumeration in terms of [[GetOwnProperty]], [[OwnPropertyKeys]], [[GetOwnProperty]]. If a virtual object really wants to synthesize some other view of inherited enumerable properties it should do it using those fundamental operations.



properties are added to the target object during enumeration, the newly added properties are not guaranteed to be processed in the active enumeration.

as appling to properties that have had their enumerable changed in addition to added/deleted properties. Approximately that same language appeared in both ES3 and ES5 as part of the normative semantics of for-in. During ES5 when we originally specified Object.defineProperty semantics our conceptual model of changing the attributes of an existing property was that it was equivalent to atomically deleting the old property and adding back the same property with revised attribute values.



Ijharb commented on Jan 29



Regarding an enumerable check, please note that as much as I love that check and think it should be included, var obj = { get a() { Object.defineProperty(this, 'b', { enumerable: false }); }, b: 42 }; for (var k in obj) { console.log(k, obj[k]); } does indeed log "b, 42" in Safari 9, Chrome Canary, Firefox Nightly, Edge, and IE 11 (and presumably all sorts of others).



tvcutsem commented on Jan 29



It took me a while to ingest all the various debates that lead to this issue, but I think I now have a better understanding of the constraints.

I agree with the counter-arguments to (2), (3) and (4). I was unaware that the iterator must be exhausted anyway for other reasons.

That leaves point (1). To make it easier to see if there would be potential issues, let's consider a concrete use case of virtualizing the proto-chain: implementing an object that pretends to support multiple inheritance or mixin-style inheritance. Obviously, getPrototypeOf will betray the illusion as it can return only 1 single object, but get(), set(), has() and enumerate() can virtualize multiple inheritance just fine. Without an enumerate() trap, I guess the properties of all prototypes can still be enumerated by having getPrototypeOf return either a synthesized "aggregate" prototype (containing the union of all inherited properties) or by synthesizing a linearized chain of prototypes. This is not as efficient or elegant as the direct enumerate() approach, but it seems like it could be made to work.

Would it make sense to leave enumerate() in but change its return type from an iterator to an array? If the iterator is exhausted and its results kept in memory anyway, specifying that the result must be an array will make that cost clear to developers. It would also ease the invariant checking.



allenwb commented on Jan 29





@tvcutsem

Without an enumerate() trap, I guess the properties of all prototypes can still be enumerated by having getPrototypeOf return either a synthesized "aggregate" prototype (containing the union of all inherited properties) or by synthesizing a linearized chain of prototypes. This is not as efficient or elegant as the direct enumerate() approach, but it seems like it could be made to work.

that's precisely what I had been assuming. And I think it is just fine. If somebody is trying to create something like a prototype that encapsulates MI lookup then they should want its synthesized inheritance model to any client that introspects on inherited properties using [[GetPrototypeOf]], not just for-in enumeration.



domenic commented on Jan 29



My take on this issue is that it (1) is indeed a reasonable point from a purity perspective. However, we have to remember what exact feature is getting the shaft here: namely, for-in. For-in is in general considered a legacy feature, so saying that it is not as easy to virtualize as get/set/has seems OK. You can still do it; you still maintain full control of the getPrototype trap, after all. You'll just have to invest a little more work if you want for-in to work consistently with your get/set/has.



tvcutsem commented on Jan 30



@allenwb agreed that a good MI Proxy implementation will need to synthesize the proto-chain anyway for [[GetPrototypeOf]] to work.

Ok, so removing <code>enumerate()</code> would work (it was never a fundamental trap anyway), but I'd still like to know whether the option of having <code>enumerate()</code> return an array rather than an iterator would solve the most pressing issues and give us the best of both. It doesn't have the problem of causing observable iterator side-effects (other than invoking the trap, which is fine), it makes the memory cost manifest, it provides the VM with all property keys at once (supporting the snapshotting semantics), and it still allows easy virtualization of inherited properties. Thoughts?



rossberg-chromium commented on Feb 2



@tvcutsem, [[Enumerate]] introduces quite a bit of complexity and weirdness. It makes the separation of responsibilities between for-in and the different [[Enumerate]] methods rather fuzzy, see e.g. bugs ecma262/#160 and #161 and associated discussions and PRs. It exposes an underspecified method in the reflection API that is really just a hack for for-in. It would simplify matters significantly (and fix all open issues almost immediately) if we just got rid of it altogether and move all hacks to for-in.



tvcutsem commented on Feb 2



@rossberg-chromium Fair enough. If removing the [[Enumerate]] internal method would significantly clean up the spec, that's also a good indicator it doesn't belong in the reflection API.



bterIson commented on Feb 2



I will work up a PR for this.

@tvcutsem thanks so much for working through this with us :-D



bterlson commented on Feb 5



Proposed fix is here: tc39/ecma262#367.



Ijharb commented on Feb 5



Linking to tc39/proposal-object-values-entries#12 as well (minor spec text update in the proposal)



allenwb commented 16 minutes ago



Is there any way to transfer this issue to ecma262 repo? It has background info that should be public.

Probably a lesson that "Reflector" issues are best for getting attention of TC39 delegates but shouldn't be used for discussions that can/should be publicly visible.

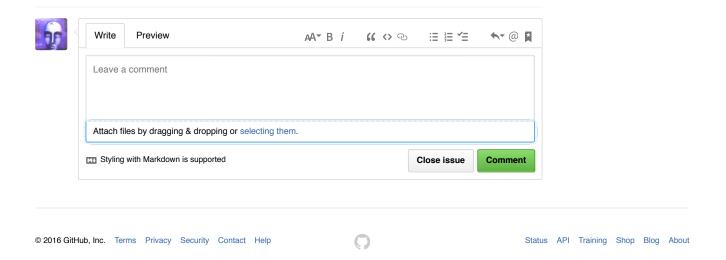


ljharb commented 10 minutes ago



The only way would be to make this repo public, afaik.

Alternatively we could laboriously reconstruct the issue thread in order, and then only the timestamps would be wrong?



7 of 7