Chrome Dev Tools deferred module namespaces support

Authors: caiolima@igalia.com
August 2026

Signed off by

Name	Approval status
Danil Somsikov (DevTools TL)	LGTM
Philip Pfaffe (Domain expert)	LGTM

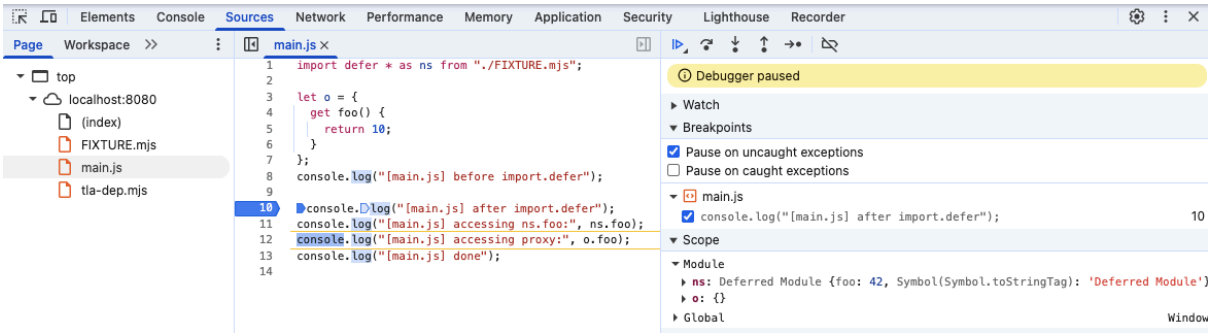
Problem

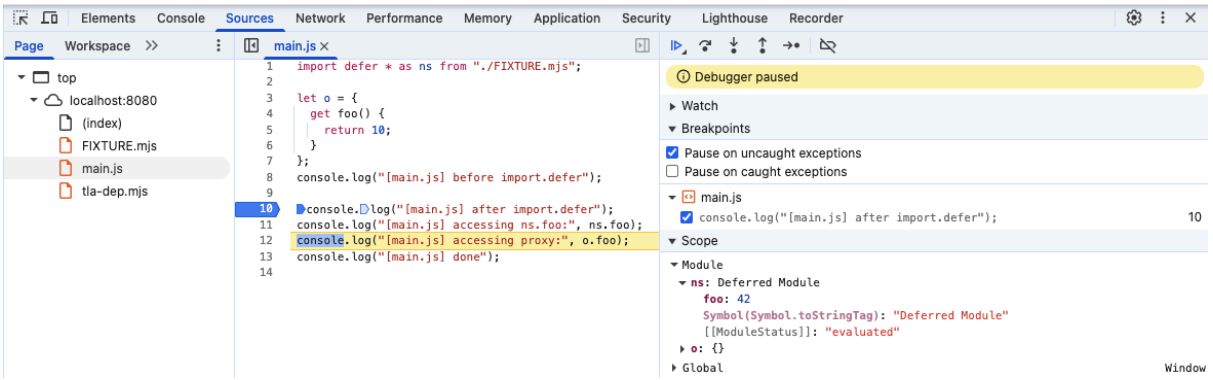
The way Chrome Dev Tools is structured now causes deferred modules to be evaluated once they are inspected. The reason for this to happen is because on Object preview on Scopes and on Console Log, the operation `.getProperties` on the object triggers deferred modules to be evaluated, because it uses `KeyAccumulator::CollectOwnKeys`. Such behavior makes debugging code with `import defer` or `import.defer` non-optimal, given it will change the behavior of the program just by enabling debugger.

Bug:

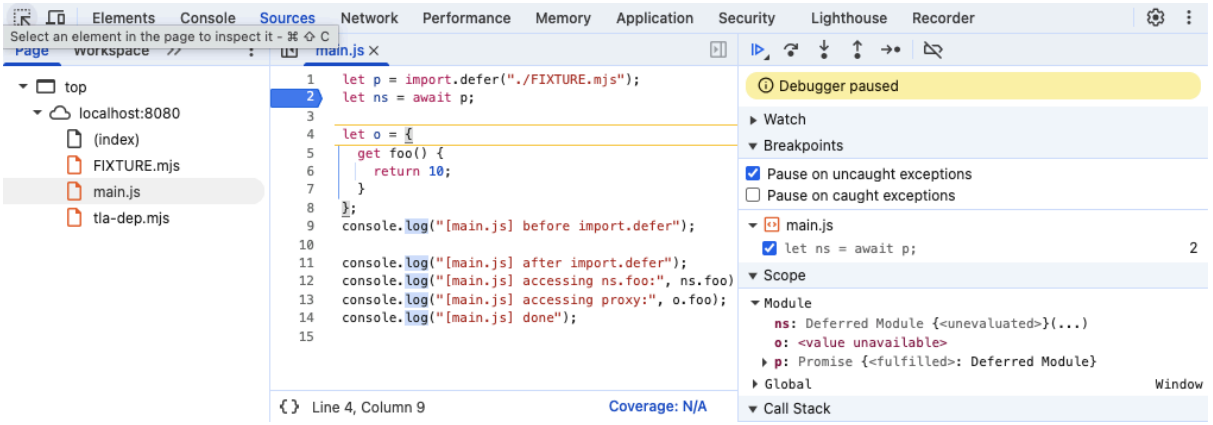
Proposed Solution

The idea to solve that is to allow Chrome Dev Tools to interact with deferred modules without triggering its evaluation by accident. To do that, the proposal is that we have a dedicated UX for deferred modules. It should show a preview if the module is evaluated, following the UX for ordinary module namespaces, like on the images below:

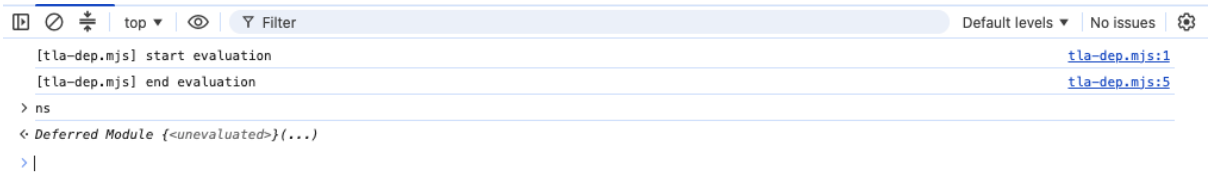




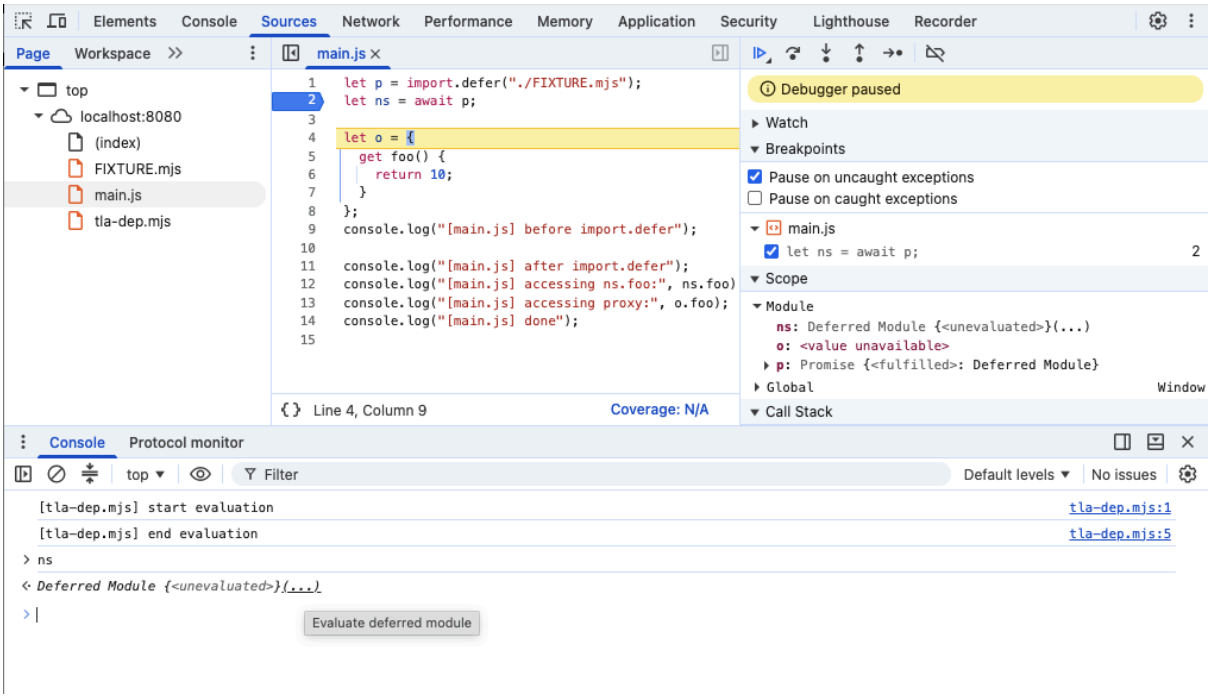
But if the module is not evaluated, we mark the in the preview that the module is not evaluated, and disallow expansion until it gets evaluated:



This also applies for preview on console:

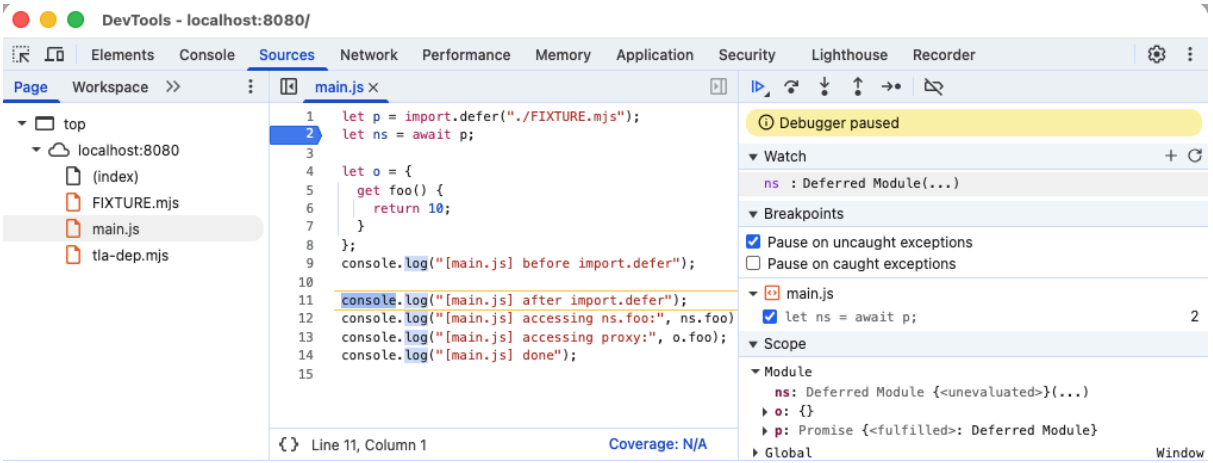


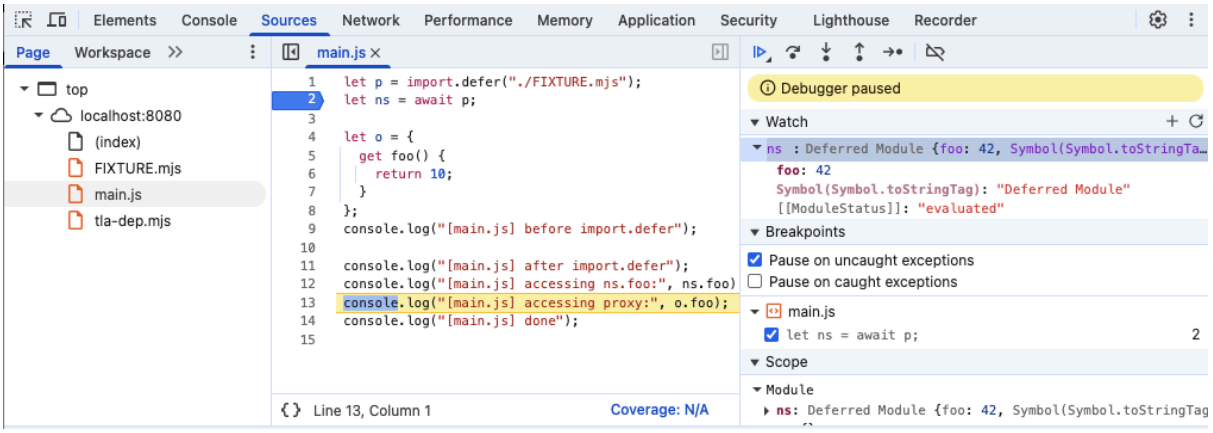
With such change, we also include a way where the user can force an evaluation, so they are able to inspect the namespace if needed. It's done by clicking in the `(...)` on preview (this mirrors getters UX):



Watchpoints

The proposal to watchpoints is to follow what we have on Scopes panel and on console preview. Once a user creates a watchpoint for `ns`, it won't trigger module evaluation, and once a module is evaluated, we properly show the preview:





Implementation details

To support this change, it'll be necessary to change both V8 and Dev Tools Frontend, with some change also on CDP.

V8 Changes

The first thing that requires a change on V8 is to not trigger evaluation of an unevaluated deferred module when it is processing `Runtime.getProperties`, even if preview is enabled. The behavior now is that it will call `KeyAccumulator::CollectOwnKeys` and since this path is shared with `[[OwnPropertyKeys]]`, this will cause the module to be evaluated. We need to add a dedicated path on V8 that if the call is coming from debugger and the deferred module is not evaluated, we skip the evaluation.
It's important to notice that there's a way to collect all export names of a module, since it's created on linking time, however we'd need a proper way to return such properties marking their values as unavailable on `PropertyPreview::TypeEnum`, but given it would only show exports without their values, I'd argue that such complexity doesn't see necessary, given the idea is to provide a way to force a module evaluation that would show exports with their values.

Another change on V8 is to expose `[[ModuleStatus]]` of those modules, so the front-end can properly show the UX for each status.

Lastly, we need to introduce a new `ObjectPreview::SubtypeEnum::Deferredmodule` so frontend can properly branch when displaying such object on Debug, Console and Watch panels. This is where we have a small change on CDP.

Frontend changes

For the Frontend side, we can split the work in two parts. The first part consumes the new information and renders a deferred namespace. The second part adds a button that forces evaluation, so we can decide to ship the first part alone.

Starting with the first part, `SDK.RemoteObject` needs to be changed to support deferred namespace objects. A deferred namespace can then be identified by its subtype being `deferrednamespace` and the status then comes from `[[ModuleStatus]]` in the object's preview, which gives us `isUnevaluatedDeferredModuleNamespace` and `isEvaluatedDeferredModuleNamespace`. Note the status is only available where a preview was requested, so panels like Watchpoint can't tell the two states apart and have to assume the module hasn't run.

With that, `RemoteObjectPreviewFormatter` can render the status inline the same way it turns `[[PromiseState]]` into `Promise {<pending>}`, so an unevaluated namespace reads `Deferred Module {<unevaluated>}` and a failed one `Deferred Module {<errored>}`. We also want to make these objects non-expandable until the module runs, which is a single override of `RemoteObject.hasChildren`, since that's the property read by `ObjectPropertyTreeElement.updateExpandable` to decide to create the component that shows expansion.

One change on `ConsoleViewMessage.formatParameter` is required by the addition of a new subtype, where it switches on `subtype || type`. `deferredmodule` has to be registered there alongside `map`, `set` and `promise`.

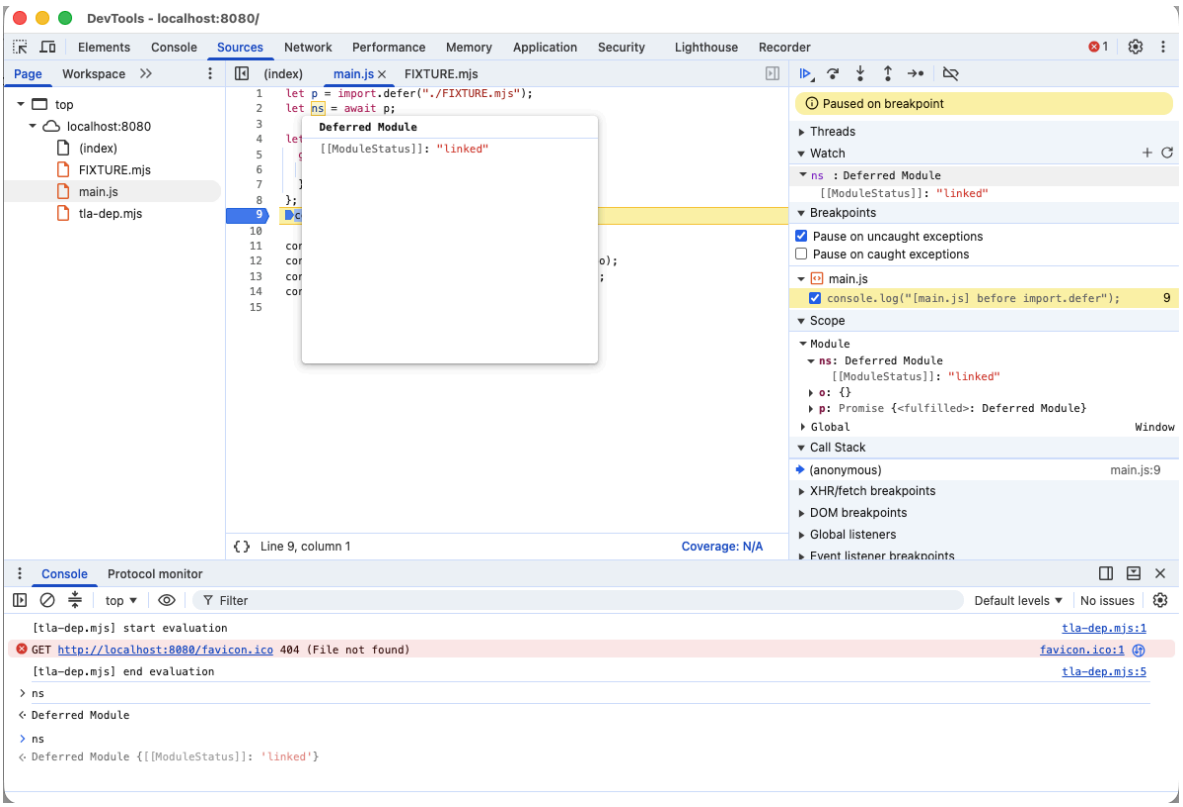
Without implementing the second part, a developer who wants to inspect the deferred module can type `ns.something` in the Console. This means that adding a button to trigger the evaluation is mostly a quality of life feature than a feature that gives a new capability to users.

The second part is the addition of the `(…)` button, which mirrors the behavior for property getters. Evaluation can be forced with a `Runtime.callFunctionOn` that accesses the namespace. That call has to pass `generatePreview`, so the returned namespace comes back with the preview that carries its new status and we can update components properly. One important complexity that this UX brings is that each component owns its own copy of the value, so the widget can't swap the value in place, and we need a `deferred-module-evaluated` event that `ObjectPropertyWidget`, `ConsoleViewMessage` and the Watch panel handle to adopt the evaluated object.

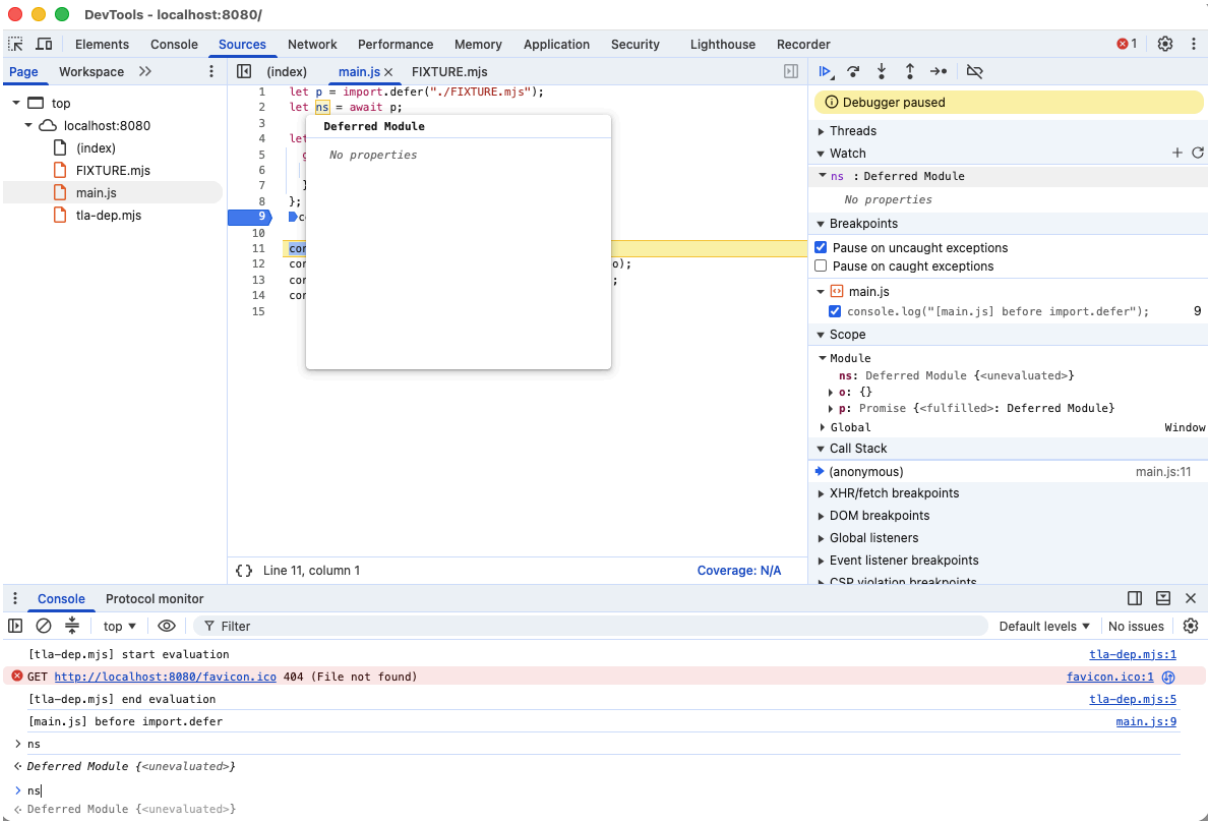
UX without any intervention on Frontend

With the introduction of V8 changes by this CL, it was possible to test with ToT devtools and verify how the experience is without any frontend intervention.

Since `preview: true` on deferred namespaces now return a new internal field, that appears as a field on deferred module namespaces:

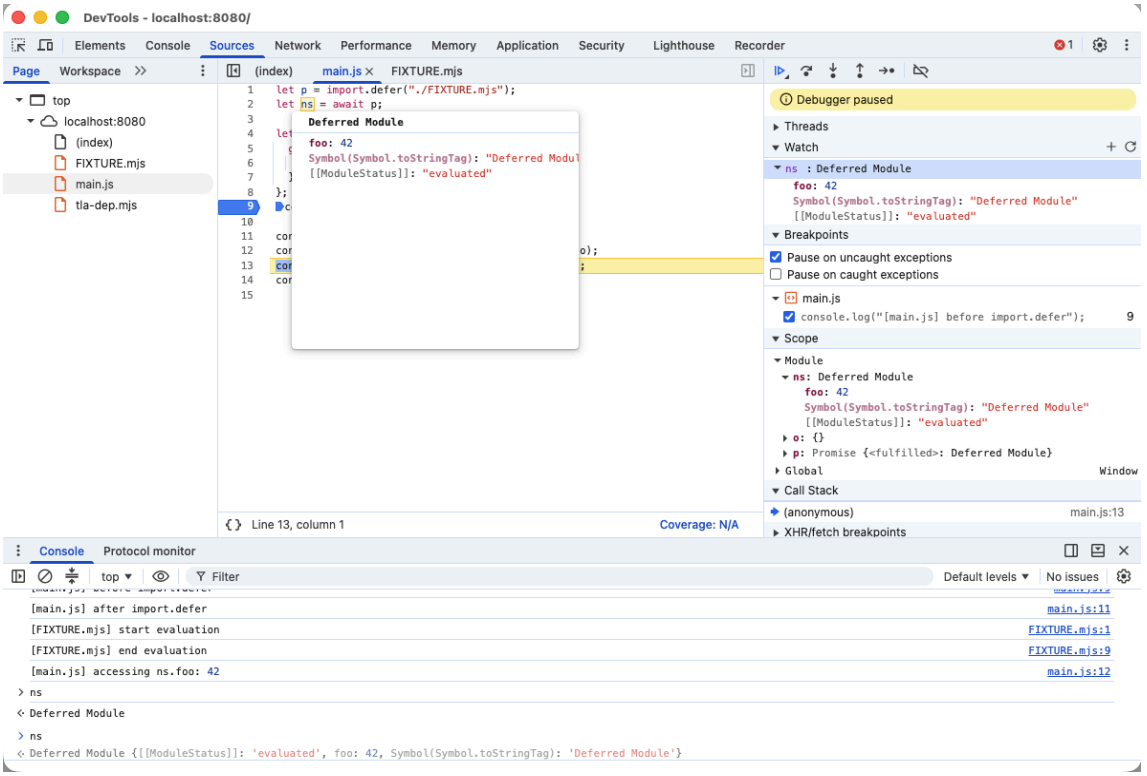


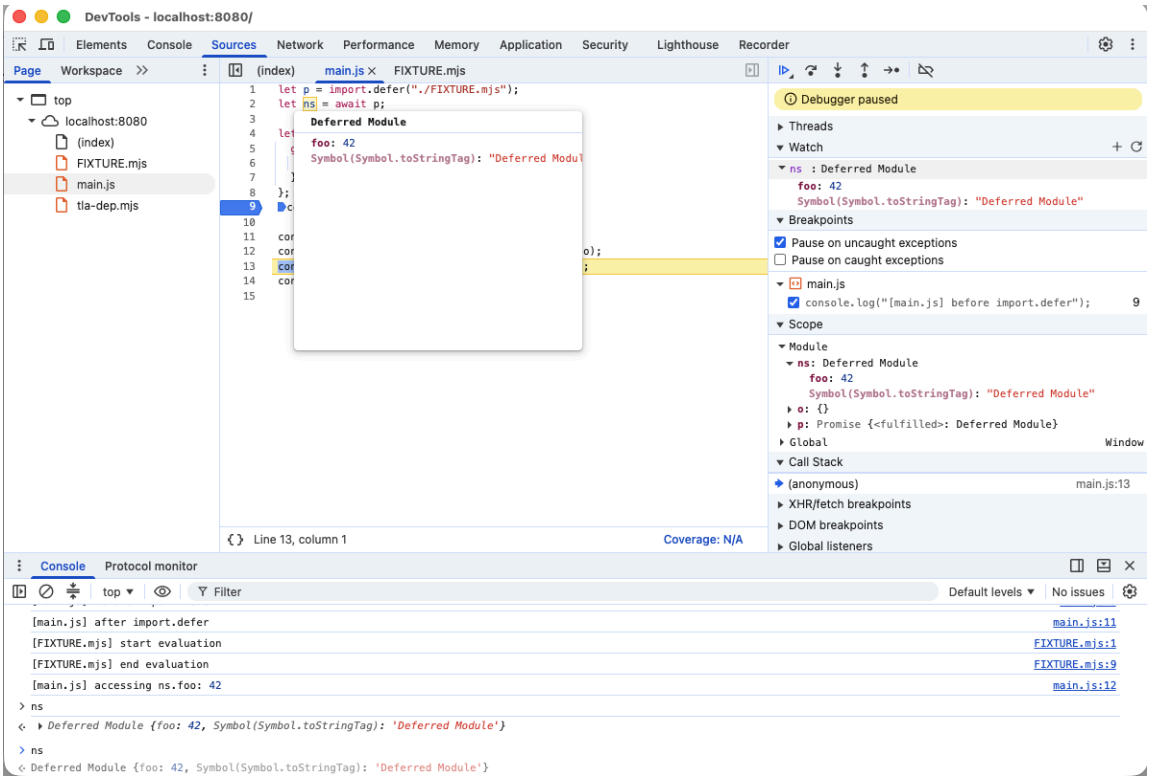
From the image above, we can see that the preview of deferred module namespace just shows `[[ModuleStatus]]`, and the description comes from `@toStringTag`. The proposed solution looks like this when the module is not evaluated:



The difference here is that on the proposed frontend implementation, `[[ModuleStatus]]` is not exposed directly and it's used to do a formatting on object description as `DeferredModule {<unevaluated>}`.

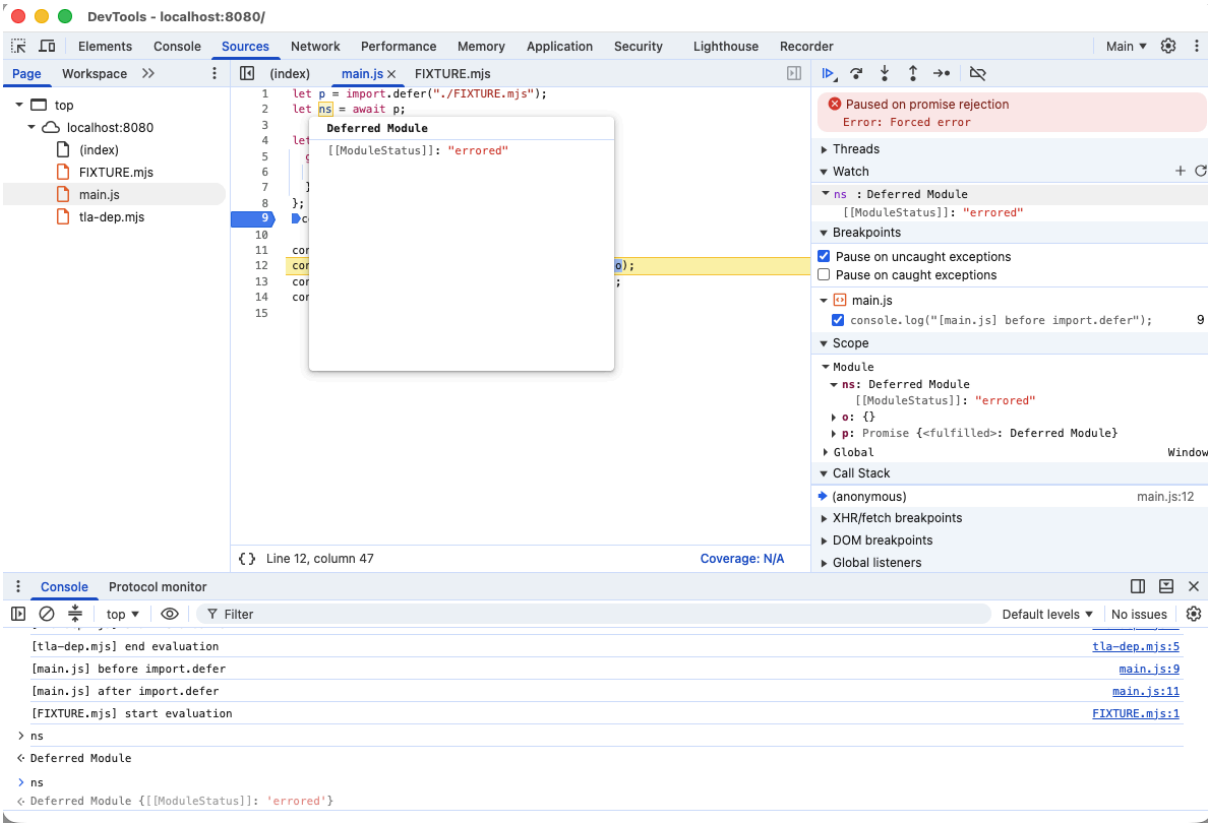
When the module gets evaluated, here's how the preview looks on preview panels (abve is ToT, bellow implements UX changes:

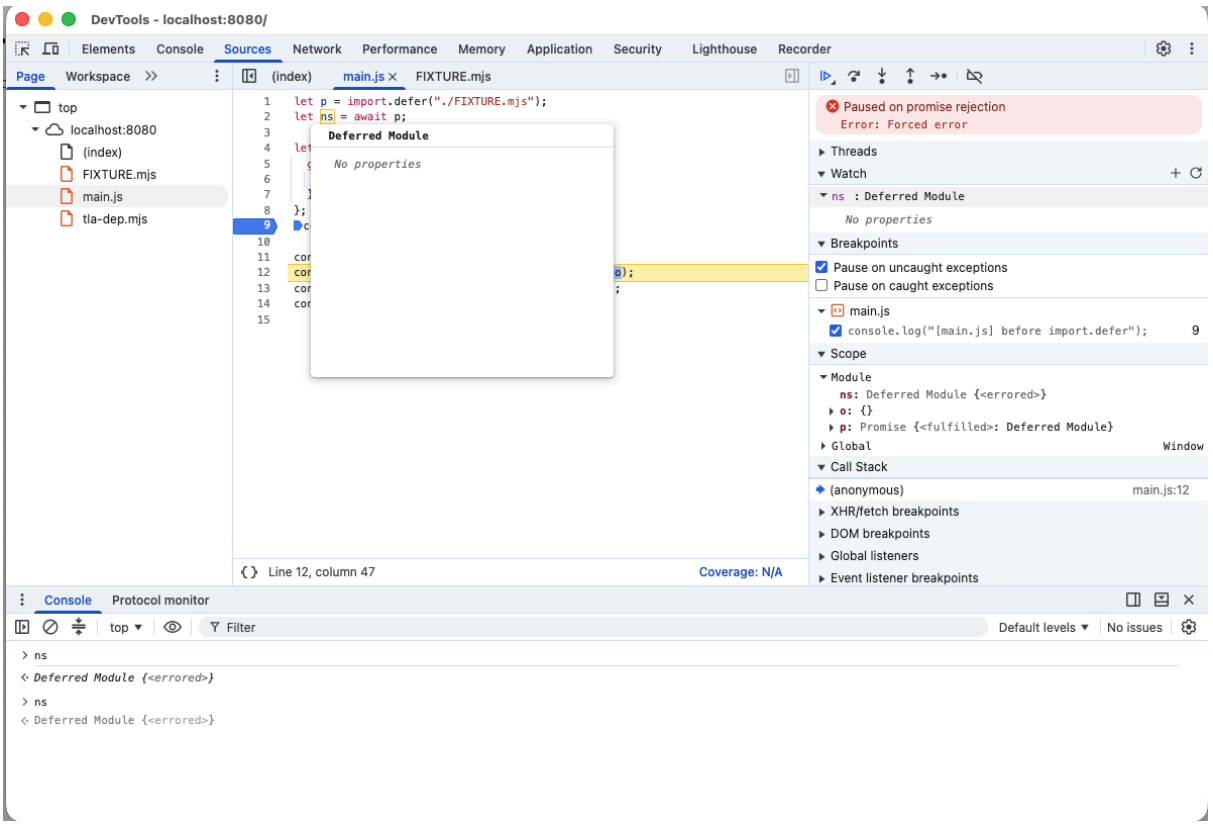




The difference there is that with a new UX `[[ModuleStatus]]` is hidden and the preview looks like an ordinary object. Also, the preview on console shows properties on tree object.

When there's an evaluation error, this is how the panels are reported (first is ToT, and bellow is with Frontend changes:





Again the difference is that the proposed ToT treats deferred namespace as an object with internal slot `[[ModuleStatus]]` exposed, while the changes proposed on frontend hide this field and changes object description.

Overall there's not much noticeable difference on UX between versions.

Tests

On V8 side we will include automated inspector tests to validate that calling `Runtime.getProperties`, `Debugger.evaluate`, `Debugger.evaluateOnCallFrame`, `Runtime.awaitPromise`, `Runtime.runScript`, and `Runtime.callFunctionOn` with `generatePreview: true` doesn't trigger the evaluation of a deferred module is not evaluated. The expected result here is that the preview of this module doesn't include any of its export names. If the deferred module is evaluated, it should return the preview as expected.

On Frontend, we will add automatic tests that will validate the presentation rules when we get a deferred module returned from operations, validating behavior for unevaluated and evaluated states.

Future work

One thing that we could consider for the future is to allow inspecting a deferred module before it gets evaluated, since exports are known before evaluation, and just their value depends on evaluation to finish. Implementing it requires some changes on CDP to properly represent properties without available values on PropertyPreview.