[PUBLIC]

Design Doc: ES6 modules in Blink

kouhei@, dominicc@
Short link to this doc: bit.ly/modules-in-blink
◂ Old doc (google internal)

Background

(TODO: proper text)
ES6 modules! In! Blink! Will be great!

Now we have V8 support for ES6 modules thanks to adamk@, we just need to have Blink implement it in order to ship.

Contributors

kouhei (Blink loading)-- fetching, script loading, resolution, tree fetch, running, bindings, parser
japhet, hiroshige (Blink loading) - script tag processing
adamk (V8) -- bindings
neis (V8)--module internals, linking, evaluation
dominicc (Blink HTML&DOM)--parser
kochi (Blink HTML&DOM) -- perf&web-platform-test testing
domenic (Blink Standards)--spec, test reviews; TC39 ...
addyo (Web DevRel)--bring web developers
seththompson (DevTools PM)--dev tools integration planning
haraken, yhirano - Blink core+bindings CL reviews

Current Status: Where are we at?

See this doc.

Implementation

Design Philosophy

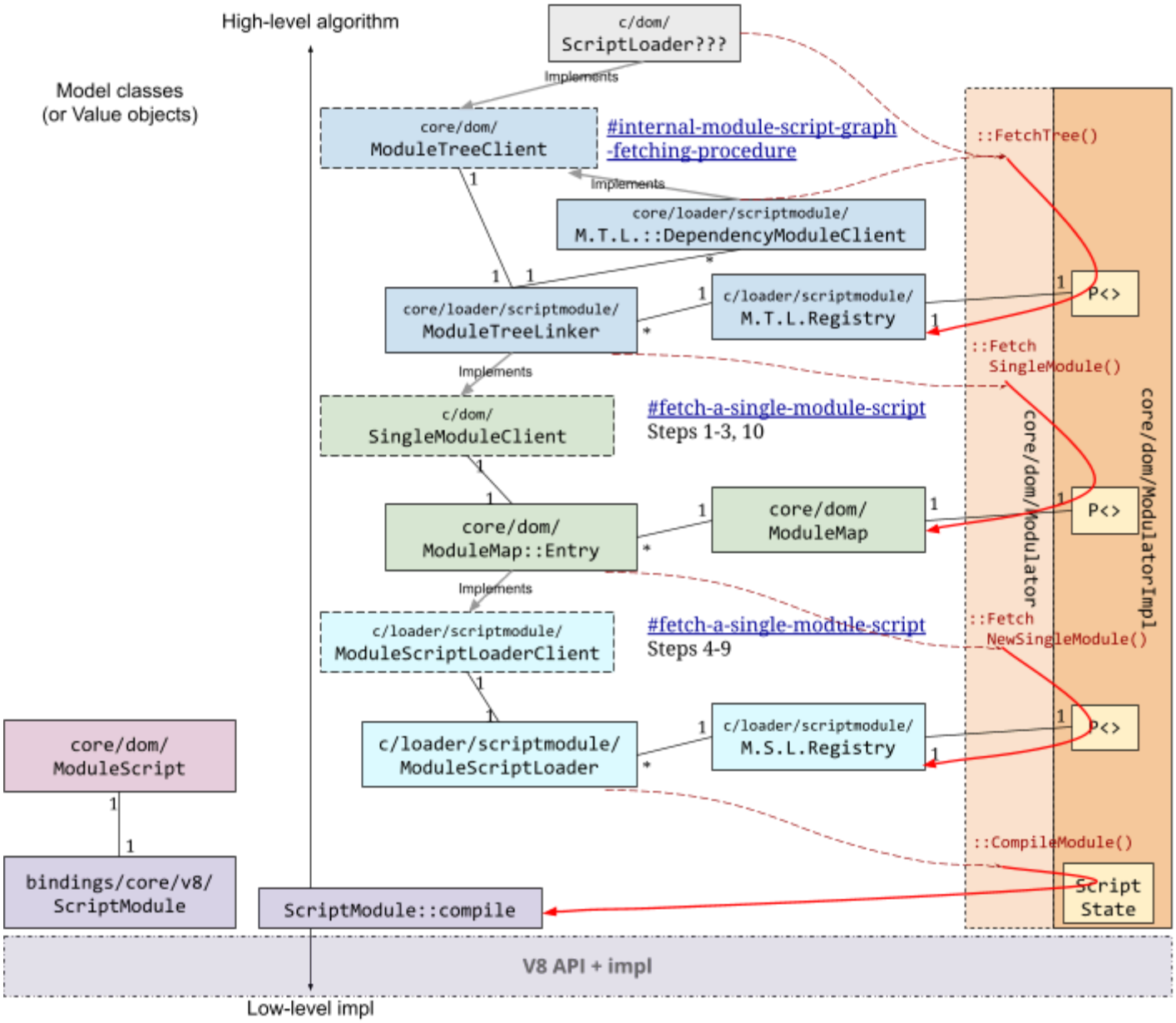
Design pattern and coding idioms of the implementation mostly match Blink style. However with some twist this time.

Highlights of diffs from “traditional” Blink code:

- Unit testable design
    - Each component is tested
    - Each unit test runs with itself + minimum dependencies.
    - Specifically, we try to avoid requiring DummyPageHolder/SimTests for everything.
- Clear correspondence to the spec text
    - Copy-n-paste spec text in comments indicates which part of the spec the code implements.
        - TODO: How can we make this sustainable to follow future spec changes?
    - Follow the spec. Upstream any optimization we have found:
        - Sub-tree instantiation complete flag to early-exit “uninstantiated inclusive descendant module scripts” finding algorithm
        - Remove intermediate list?
- Small classes
    - Responsibility of each class is clearly defined + fine scoped.
        - A single sentence should be sufficient for explaining what a class does.
    - Typically a class corresponds to a single spec algorithm/concept.

Overview

Fetching a module tree



See also: [Spec↔Implementation mappings and arguments flow]

ModuleTreeLinker runs “internal module script graph fetching procedure” asynchronously, and notifies ModuleTreeClient when done. ModuleTreeLinkerRegistry class prevents active instances of MTC from GCed.

ModuleScriptLoader runs “fetch a single module script” procedure asynchronously and notifies ModuleScriptLoaderClient when done. ModuleScriptLoaderRegistry class prevents active instances of MSL from GCed.

Error Handling

V8 Module implementation can report failures from specced algorithms. How should the error info flow?

- ParseModule (ScriptModule::compile)
  - Consumer:
    - Only Console.log
  - We can handle it just like classic script parse failure reporting path.
  - v8::TryCatch setVerbose(true) -> global message handler (V8Initializer::messageHandlerInMainThread)
- ModuleDeclarationInstantiation, v8::Module::Instantiate, ScriptModule::Instantiate
  - Emitter (V8 impl):
    - Can throw SyntaxError, etc.
    - Domenic: “I guess technically this can only be one of the errors given by https://tc39.github.io/ecma262/#sec-moduledeclarationinstantiation, so SyntaxError, TypeError (via HostResolveImportedModule), maybe a few more.”
  - Consumers:
    - “internal module tree fetching proc” stores the error to”module script”.”instantiation error”
    - When “running a module script”, we need to report the stored exception.
    - HostResolveImportedModule need to rethrow the stored exception when the resolved module has error state.
- Module run
  - *Not investigated yet. Same as classic script run?*

Implementing “instantiation error” concept in Blink bindings is a bit tricky.
From the usage, we can derive the following requirements:

- Blink can store the “Instantiation error” info on heap.
- “Instantiation error” info extract from V8 exception handler.
- Blink can report the “Instantiation error” to console.
- “Instantiation error” can be re-throwed as V8 exception.

We have explored several implementation options which meets the requirements:

1. Store the v8::Exception as ScriptValue.
   a. This is risky as the value may hold strongrefs to any objects in V8 heap, and cause memory leak.
   b. “Instantiation error” dies when “module map” dies, which == lifetime of the v8::Context.

2. ExceptionState
    a. Won't work, as this is STACK_ALLOCATED
3. DOMException
    a. Can be stored on Blink heap.
    b. Currently cannot be constructed from V8 exception.
        i. We have v8::TryCatch.Exception() to access exception v8::Value.
        ii. However, the V8 embedder (Blink) can't extract the typeinfo (ideally V8ErrorType)
    c. Can report the instantiation error to console.
    d. Currently cannot be re-throwed as V8 exception w/ its typeinfo maintained (SyntaxError, TypeError, etc.)
        i. However supporting V8TypeError as ExceptionCode is trivial.

Update: now resolved. yhirano@, haraken@ and kouhei@ agreed to keep m_instantiationError as TraceWrapperMember<v8::Value>

Open Questions (Implementation)

How should the inline module scripts be handled?

hiroshige@ solved this ->

- Each inline module script will have a ModuleScript instance.
- Inline module script would not go through ModuleMap layer. Inline module script would not have a module map entry.

Web Developers

Tools

A bunch of things should "just work" through v8::Source, resource fetching, etc. but will need tests.
Some use cases which may need more work:
- Script debugging: Jump from import to definition.
- Failed import loading debugging: Find the status of a script tag, find out why it failed.
- Draw graph of dependencies; getting a sense of how well-factored your app is

Documentation

We need a best practices document for loading ES modules "as scalably as possible with current technology." Scalable loading will deliver better solutions later.

Test Plan

Cover these cases.

Shadow DOM and Custom Elements were slow (are slow?) to upstream their test cases. This time we'll cover the implementation with C++ unit tests on a per-change basis to decouple coverage and web platform test upstream/dedup/import.

Scalable Loading

The primary question regarding scalable loading is whether fetching and parsing/evaluating can be balanced. This is at the expense of determinism so it is not an obvious choice. See JS Modules: Determinism vs ASAP (chromium.org sign-in.)

Delivering (packaging, compressing, etc.) the multiple separate module script files is a separate issue. We think this is not particularly different to other blocking/dependent resources like classic scripts, stylesheets, etc.

FAQs

What happens if a script does "import"? It's a syntax error--only modules can import stuff from other modules. ES has a top-level "Module" production which is how modules are parsed.

So how does the first module get loaded? There are two ways: 1. <script type="module"> tag. 2. A method of the WHATWG Loader spec, such as "import". The WHATWG Loader spec has a large surface area and is still pretty rough, so we'll probably ship script tags first.

What happens if there's a redirect fetching a module? See the note in step 10 of fetch a module script: The module map is keyed by request URL, but the module script's base URL is the response URL. So fetches are de-duped by request URL and module URLs are resolved by response URL.

References

- JS Modules: Determism vs. ASAP (chromium.org sign-in)
- Scalable Loading Project
- Blink Loader "Yukari" Code Organization
- ECMA 262, Imports, Scripts and Modules
- HTML, script type=module
- WHATWG, Loader
- Modules in V8
- ES Modules in Blink RL Update--this was aborted due to schedule mismanagement
- Kouhei's first cut patch