The V8 public C++ API

Overview

The V8 public C++ API aims to support four use cases:

1. Enable applications that embed V8 (called the embedder) to configure and run one or more instances of V8.
2. Expose ECMAScript-like capabilities to the embedder.
3. Enable the embedder to interact with ECMAScript by exposing API objects.
4. Provide access to the V8 debugger (inspector).

Configuring and running an instance of V8

V8 requires access to certain OS-level primitives such as the ability to schedule work on threads, or allocate memory.

The embedder can define how to access those primitives via the v8::Platform interface. While V8 bundles a basic implementation, embedders are highly encouraged to implement v8::Platform themselves.

Currently, the v8::ArrayBuffer::Allocator is passed to the v8::Isolate factory method, however, conceptually it should also be part of the v8::Platform since all instances of V8 should share one allocator.

Once the v8::Platform is configured, an v8::Isolate can be created. All further interactions with V8 should explicitly reference the v8::Isolate they refer to. All API methods should eventually take an v8::Isolate parameter.

When a given instance of V8 is no longer needed, it can be destroyed by disposing the respective v8::Isolate. If the embedder wishes to free all memory associated with the v8::Isolate, it has to first clear all global handles associated with that v8::Isolate.

ECMAScript-like capabilities

In general, the C++ API shouldn't enable capabilities that aren't available to scripts running in V8. Experience has shown that it's not possible to maintain such API methods in the long term. However, capabilities also available to scripts, i.e., ones that are defined in the ECMAScript standard are there to stay, and we can safely expose them to embedders.

The C++ API should also be pleasant to use, and not require learning new paradigms. Similarly to how the API exposed to scripts aims to provide good ergonomics, we should aim to provide a reasonable developer experience for this API surface.

ECMAScript makes heavy use of exceptions, however, V8's C++ code doesn't use C++ exceptions. Therefore, all API methods that can throw exceptions should indicate so by returning a v8::Maybe<> or v8::MaybeLocal<> result, and by taking a v8::Local<v8::Context> parameter that indicates in which context a possible exception should be thrown.

API objects

V8 allows embedders to define special objects that expose additional capabilities and APIs to scripts. The most prominent example is exposing the HTML DOM in Blink. Other examples are e.g. node.js. It is less clear what kind of capabilities we want to expose via this API surface. As a rule of thumb, we want to expose operations as defined in the WebIDL and HTML spec: we assume that those requirements are somewhat stable, and that they are a superset of the requirements of other embedders including node.js.

Ideally, the API surfaces defined in those specs hook into the ECMAScript spec which in turn guarantees long-term stability of the API.

The V8 inspector

All debugging capabilities of V8 should be exposed via the inspector protocol. The exception to this are profiling features exposed via v8-profiler.h. Changes to the inspector protocol need to ensure backwards compatibility and commitment to maintain.