Author: jochen@
Last modified: 2014-09-17

V8 Exception Handling in Blink

Summary

Exception handling in blink is currently broken on many levels. This document aims to list all instances of exception related problems, and identify individual projects.

Background

Exceptions in V8 can be triggered either by JavaScript during script execution, or script execution during C++ API calls (e.g. v8::Object::ToString), or by certain API methods (e.g. v8::Isolate::RequestTermination).

Depending on the kind of exception, exceptions can be caught by try/catch blocks in the script, using C++ TryCatch objects, message handlers, or combinations thereof. Most notably, the TerminateExecution exception cannot be caught by script.

If an uncaught exception is pending, API methods that can run JavaScript will return NULL pointers. This results in code like this or this and bugs like this and this.

Goal

The goal is to change the exception API and its usage in blink such that it is more difficult / impossible to introduce the kind of bugs we see today.

Problems

This is a list of all issues with exception handling I currently know of.

Execution termination

When an embedder requests termination, and uncatchable oddball exception is thrown. V8 returns immediately to the embedder and all further API calls will return immediately and return invalid results (e.g. empty handles, NULL pointer).

This can lead to problems when V8 was terminated while there are C++ callbacks on the stack. Typically blink bindings do not check for invalid return values from V8 during callbacks.

This happens when stopping a worker, or when stopping script execution in an Android WebView. In the latter case, the code path is disabled, because crashing the single-process WebView would terminate the entire application.

Uncaught exceptions

If exceptions happen outside of JavaScript execution and without a C++ TryCatch object on the stack, the exception is stored on the Isolate and all further C++ API calls will fail returning invalid results.

Since exceptions can be thrown in any API call (e.g. caused by a stack overflow), every API call would have to be protected by a TryCatch object or by using ExceptionStates, but even then it is unclear what should happen with the exception.

Potential Solutions

- Stack overflow exceptions during C++ API calls should lead to an immediate crash. This is consistent with the behavior of C++ code.
- C++ API calls that can throw exceptions should crash if they are invoked without a TryCatch block on the stack or outside of script execution.
- Unclear how to solve stack unwinding issue. Maybe introduce MaybeHandle<> in the V8 API. Rewriting all the handles in chromium & blink again, anybody? Or just audit the bindings generator?
- haraken@: For the termination issue, another idea would be to implement TerminationForbiddenScope and add the scope to all DOM attributes/methods called in workers? That way we can guarantee that V8 APIs called inside DOM attributes/methods in the scope don't return empty handles. Given that DOM attributes/methods are guaranteed to be complete in a finite time, it would be acceptable to forbid the termination signal while executing the DOM attributes/methods.
- mnaganov@: How about emulating C++ exceptions with setjmp / longjmp? So we can bail out from a block of code that interacts with V8, like this:

  ```
  BEGIN_TRY_CATCH(v8::TryCatch try_catch)
  v8::Handle<...> foo = v8::foo();
  v8::Handle<...> bar = v8::bar();
  …
  CATCH
  // we fall down here if any of the operations above has failed,
  // here we can examine |try_catch| to see, what has happened
  END_TRY_CATCH
  // continue execution
  ```

  Pros:
  - no need to check, if the returned handle is empty after every call to V8;

But I’m not sure about all the consequences of this approach.