Remove Arguments Adaptor Frame in Deoptimizer

Attention: Shared Google-externally

Author: victorgomes@
Tracking Bug: `v8:10201`
Status: Done

Background

We are removing the arguments adaptor frame, used when there is a mismatch between the actual number of arguments and the formal parameter count, see the design doc. This document explores the changes needed to be done in the deoptimizer.

The Deoptimizer

The deoptimizer is responsible to reconstruct all the non-optimized frames (it can be more than one in case of inlining). It reconstruct 4 possible frames:

- Interpreted Frames
- ArgumentAdaptor Frame
- BuiltinContinuation Frame
- ConstructStub Frame

During compilation, TurboFan creates code metadata (TranslatedFrames and TranslatedValues) that guide the reconstruction of these frames during deoptimization.

What needs to be done

If we eliminate the argument adaptor frame, the deoptimizer will need to:

1. Push the extra arguments.
2. Push the actual arguments count in the correct interpreted frame slot.

Push the extra arguments

Nothing needs to be done for ConstructStub and BuiltinContinuation frames, since they work without an adaptor frame. They already contain the number of arguments and a copy of each argument (see frame-constants.h and the builtins that create them [1, 2]). The interpreted frame, however, does not know how many arguments were pushed in the stack. And in case of inline frames, it relies on the reconstruction of an adaptor frame. TurboFan creates a FrameState node that contains this information.

Push the actual argument count

As I mentioned, the interpreted frame does not know how many arguments were pushed. If the deoptimizer is reconstructing the bottom most frame, it can use the information provided by the optimized frame. But in cases of inlined frames, the deoptimizer currently does not have the actual argument count.

Solution #1: extra metadata in the FrameState

A possible solution is to incorporate more metadata in the InterpretedFrameState. We would need to distinguish two cases, either it is the outermost frame state or an inline frame.

In the case of outermost frame state, the compiler cannot know how many arguments will be used to call the optimized code. The deoptimizer will simply use the arguments count provided by the optimized code frame. The extra arguments will not need to be pushed, since it should be already on the stack. Arguments with index below the formal parameter count will still need to be pushed (or even materialized), since the optimized code might have clobberred them.

In case of inlined frame state, the compiler knows how many arguments will be pushed. It should record the count and all the extra arguments in the frame state node, instead of creating an arguments adaptor frame state. Here lies a tricky question: which frame state should contain this information? In the caller or in the callee? Usually, arguments should be part of the caller, but what does it mean to attach this information to the caller frame state? What happens if the caller calls more than one function? The actual argument count in V8 is actually in the callee frame.

This solution also breaks any optimization in TurboFan that uses the AgumentsAdaptorFrameState. Note that these optimizations are only used in case of inlined functions, since the FrameState is otherwise never introduced in the TF graph.

Solution #2: an ArgumentsAdaptor virtual FrameState

The second solution is to continue to have the ArgumensAdaptorFrameState. Although we no longer have an ArgumentsAdaptor frame in the runtime, it is useful to compile inlined functions as if we did, emitting a virtual (fake?) frame state to the deoptimizer. This frame state contains the TranslatedValues for all the arguments and the actual argument count. When reconstructing the frames in the deoptimizer, it will just simply push the arguments. A WIP CL for this solution can be found here (check DoComputeArgumentsAdaptorFrame).

This solution does not break any of the inlined optimizations in TF.

Extra: what happens to the arguments object?

In the case of the bottom most frame, the arguments object can be reconstructed during deoptimization by accessing the arguments directly from the stack. For inlined frames, the object is reconstructed by materializing the elements from the ArgumentsAdaptorFrameState.