Revisiting `exnref`

Wasm CG Presentation 2023-08-01

Heejin Ahn, Ryan Hunt, Ben L. Titzer

Exception Handling Proposal History

- EH proposal has gone through several design iterations
- Originally did not include **`exnref`**
- Exnref originally introduced for several reasons
- Later removed from proposal
  - Concerns over ref-counted or GC'd exref package
  - Future compatibility with 2-phase exception handling
  - Not enough eyes/stakeholders involved in **`exnref`** removal
- Proposal moved to Phase 3 shortly after exnref removed
  - Primarily due to maturity of toolchains and engines

Motivation for Reintroducing `exnref`

- We encountered issues encountered after Phase 3
  - Difficulty in handling the identity of thrown exceptions in the JS API spec.
  - Complexity in the formal specification and engines.
  - Inability to CPS-transform (e.g. asyncify) Wasm functions with exceptions.
  - Lack of **`exnref`** significantly restricts toolchains' code transformations and sometimes leads to unnecessary duplication. (Specific feedback from J2CL authors).
- Proximate cause: new form of storage in the form of lexical rethrow
  - Unlike other existing forms of storage in Wasm

Implementation (Web) Reality

- All three browsers have shipped EH in its current form
- Important web properties already shipping EH binaries in the wild
- Toolchain support is already mature
- New languages targeting EH feature

Design constraints

- Cannot break the web
  - Shipping binaries need to continue working for some amount of time, regardless of any changes made now => binary workaround
- Limited appetite for extended design exploration/discussion
- Engines will have to support both Phase 3 binaries plus any additional adjustments for (interim) period of time
- Binary rewrite significantly improves chance of successfully deprecating any part of Phase 3
- Must support CPS transform / asyncify
- Must *document* web reality

Ways Forward

- Key stakeholders investigated feasibility of reintroducing **`exnref`**
- Two strategies have been identified that meet constraints
  - Option A: minimal delta to Phase 3 to introduce **`exnref`**
  - Option B: most consistent with existing Wasm spec, including **`exnref`**
- Browser vendors have signaled willingness to make EH proposal changes in products
- Toolchain authors (LLVM and Binaryen) have signaled willingness to support exnref
- Reasonable and clean strategies identified for formal specification for option A or B

Option A

- `try <instr>* (catch <tag> <instr>*)* (catch_all <instr>*)? end`
- Similar to the current proposal
- `catch` pushes (extracted values, `exnref`) to the stack

```
try
  …
catch $tag0  ;; pushes (extracted values, exnref) to the stack
  …
catch $tag1
 …
catch_all
 …
end
```

Option B: `try ~ catch`

- `try <instr>* catch (<tag> <label>)* (catch_all <label>)?`
- `catch` takes a list of (tag, label) and a single (optional) `catch_all` label, and branches to the corresponding label with an `exnref` value if an exception is caught, which later can be extracted with br_on_exn
- No end instruction

```
block $label0 (result exnref)
  block $label1 (result exnref)
    block $label_catch_all (result exnref)
      try
        …
      catch ($tag0, $label0), ($tag1, $label1), …, $label_catch_all
    end
  end
end
```

Option B: br_on_exn

- Extract an exnref with br_on_exn

```
block $label0 (result …)
  block $label1 (result …)
    br_on_exn $tag0 $label0
    br_on_exn $tag1 $label1
    …
  end
end
```

Common to Option A & B

- `rethrow` takes an `exnref` instead of a label (= immediate)
- Does not depend on the lexical surroundings

Community Discussion

- Should we move forward to somehow add **`exnref`** to EH?
- Feasibility of adjusting EH in a way that deprecates in-the-wild features?

Further Discussions

https://github.com/WebAssembly/exception-handling/issues/280