serde_v8

Author: Aaron O'Mullan aaron.omullan@gmail.com

Serde support for encoding/decoding (rusty_)v8 values.

Broadly `serde_v8` aims to provide an expressive but ~maximally efficient encoding layer to biject rust & v8/js values. It's a core component of deno's op-layer and is used to encode/decode all non-buffer values.

Original issue: denoland/deno#9540

Quickstart

`serde_v8` fits naturally into the serde ecosystem, so if you've already used `serde` or `serde_json`, `serde_v8` 's API should be very familiar.

`serde_v8` exposes two key-functions:

- `to_v8` : maps `rust->v8` , similar to `serde_json::to_string` , ...
- `from_v8` : maps `v8->rust` , similar to `serde_json::from_str` , ...

Best practices

Whilst `serde_v8` is compatible with `serde_json::Value` it's important to keep in mind that `serde_json::Value` is essentially a loosely-typed value (think nested HashMaps), so when writing ops we recommend directly using rust structs/tuples or primitives, since mapping to `serde_json::Value` will add extra overhead and result in slower ops.

I also recommend avoiding unecessary "wrappers", if your op takes a single-keyed struct, consider unwrapping that as a plain value unless you plan to add fields in the near-future.

Instead of returning "nothing" via `Ok(json!({}))` , change your return type to rust's unit type `()` and returning `Ok(())` , `serde_v8` will efficiently encode that as a JS `null` .

Advanced features

If you need to mix rust & v8 values in structs/tuples, you can use the special `serde_v8::Value` type, which will passthrough the original v8 value untouched when encoding/decoding.

TODO

- [ ] Experiment with KeyCache to optimize struct keys
- [ ] Experiment with external v8 strings
- [ ] Explore using json-stringifier.cc's fast-paths for arrays
- [ ] Improve tests to test parity with `serde_json` (should be mostly interchangeable)
- [ ] Consider a `Payload` type that's deserializable by itself (holds scope & value)
- [ ] Ensure we return errors instead of panicking on `.unwrap()` s