NativeIO

Performant and generic storage for the web

Emanuel Krivoy, fivedots@chromium.org

What are we trying to solve?

Challenge

Give developers a storage primitive that has similar performance and flexibility to native, with all the goodness of the web.

Use cases

- Performant SQLite, LevelDB, etc. for the web
- Fast and persistent filesystem for WebAssembly
- Fine grained control over large data through storage primitives

How does it look?

Filesystem operations

```
interface NativeIO {
    FileHandle open(String name);

    void delete(String name);

    void rename(String oldName, String newName);

    List<String> getAll();
};
```

File operations

```
interface FileHandle {
    int read(SharedArrayBuffer buffer, int offset);

    int write(SharedArrayBuffer buffer, int offset);

    void setLength(int length);

    int getLength();

    void flush();

    void close();
};
```

Some examples

Get all and rename

```
var hello = nativeIO.open("hello");
var world = nativeIO.open("world");
// Returns [“hello”, “world”]
nativeIO.getAll();

hello.close();
nativeIO.rename("hello", "small");
// Returns [“small”, “world”]
nativeIO.getAll();
```

Write and read

```
var handle = nativeIO.open("foo");
// Simplified from SharedArrayBuffer
var writeBuffer = [0, 1, 0];
// Returns 3, the number of bytes written
handle.write(writeBuffer, 0);
handle.flush();

var readBuffer = [0, 0];
// Returns 2, the number of bytes read
handle.read(readBuffer, 1);
// readBuffer -> [1, 0]
```

State of the project

Now

- Enthusiastic partner feedback
- Prototype available in Chrome
- Emscripten filesystem makes trying it out easier
- Benchmarks to compare with legacy storage and sync vs. async

Open questions

- What are the right benchmarks to verify performance?
- What are other use cases that might benefit?
- Where can we improve the API surface?

Thanks!

•••

If you have any comments or questions please reach to us in our Discourse or explainer

Appendix: Use cases in more detail

- We’ve actually ported SQLite and LevelDB to validate our API. Libraries could be distributed as Wasm modules
- Managing memory consumption by swapping active/inactive segments of data between memory and NativeIO
- Caching large assets for future sessions, with full control of access

Appendix: Sync vs. async

<u>Context</u>

- WebAssembly has issues suspending and resuming while handling asynchronous calls
- Technologies like Asyncify solve this at a performance cost
- We ran an SQLite benchmark measure the slowdown

<u>Benchmark results</u>

- Async version was overall ~3 times as slow
- Significantly slower (~30x) while reading and writing
- More research needed to validate results and pinpoint the cause