Immutable ArrayBuffers for stage 2

Mark S. Miller Agoric

Peter Hoddie moddable

Richard Gibson Agoric

Jack-Works

105th Plenary

December 2024

TC 39

Recap: Proposed ArrayBuffer API

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
resize(len: number) :void
slice(start?: number, end?: number) :ArrayBuffer
transferToImmutable() :ArrayBuffer
get immutable: boolean
get detached: boolean
get resizable: boolean
get byteLength: number
get maxByteLength: number
```

Recap: Immutable ArrayBuffer Flavor

~~transfer(len?: number) :ArrayBuffer~~
~~transferToFixedLength(len?: number) :ArrayBuffer~~
~~resize(len: number) :void~~
slice(start?: number, end?: number) :ArrayBuffer
~~transferToImmutable() :ArrayBuffer~~
get immutable: true
get detached: false
get resizable: false
get byteLength: number
get maxByteLength: same number

Status Update

- Many private positive comments
- No negative comments or objections
- Stage 1 spec text already stage 2 quality
- Moddable XS implementation in progress
- Progress on open questions…

Open: An optional length parameter?

Given

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
```

do we want

```
transferToImmutable() :ArrayBuffer
```

or

```
transferToImmutable(len?: number) :ArrayBuffer
```

?

Open: An optional length parameter?

Given

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
```

do we want

```
transferToImmutable() :ArrayBuffer
```

or

```
transferToImmutable(len?: number) :ArrayBuffer
```

?

We mildly prefer the first.

Open: zero-copy slice?

Given

```
slice(start?: number, end?: number) :ArrayBuffer
```

and

```
transferToImmutable() :ArrayBuffer
```

should we add

```
sliceToImmutable(start?: number, end?: number) :ArrayBuffer
```

?

Open: zero-copy slice?

Given

```
slice(start?: number, end?: number) :ArrayBuffer
```

and

```
transferToImmutable() :ArrayBuffer
```

should we add

<u>`sliceToImmutable`</u>`(start?: number, end?: number) :ArrayBuffer`

?

Yes.

Open: throw, or silently do nothing?

Should trying to write data in an immutable ArrayBuffer via a TypedArray element set throw, even though trying to write out-of-bounds or to a detached ArrayBuffer does not?

Should TypedArray write methods (copyWithin, fill, reverse, set, etc.) throw when their backing ArrayBuffer is immutable but the targeted range is zero-length? If so, how early or late in the algorithm? The methods currently inspect arguments after ValidateTypedArray.

How early or late in SetViewValue against an immutable ArrayBuffer should an exception be thrown? It currently inspects arguments before IsViewOutOfBounds.

Likewise for abstract operations such as ArrayBufferCopyAndDetach (which currently checks IsSharedArrayBuffer, then newLength, then IsDetachedBuffer).

And also for Atomics functions.

Open: throw, or silently do nothing?

Should trying to write data in an immutable ArrayBuffer via a TypedArray element set throw, even though trying to write out-of-bounds or to a detached ArrayBuffer does not?

Should TypedArray write methods (copyWithin, fill, reverse, set, etc.) throw when their backing ArrayBuffer is immutable but the targeted range is zero-length? If so, how early or late in the algorithm? The methods currently inspect arguments after ValidateTypedArray.

How early or late in SetViewValue against an immutable ArrayBuffer should an exception be thrown? It currently inspects arguments before IsViewOutOfBounds.

Likewise for abstract operations such as ArrayBufferCopyAndDetach (which currently checks IsSharedArrayBuffer, then newLength, then IsDetachedBuffer).

And also for Atomics functions.

Drive by implementor feedback.

But when in doubt, throw.

Stage 2?

Backup Slides from Stage 1 talk

Need immutable bulk binary data

(stage 1 problem statement)

Need immutable bulk binary data

(stage 1 problem statement)

- Embedded JS ROM
  tc53 / Moddable XS

Need immutable bulk binary data

(stage 1 problem statement)

- Embedded JS ROM
  tc53 / Moddable XS
- Defensive copying

Need immutable bulk binary data

(stage 1 problem statement)

- Embedded JS ROM
  tc53 / Moddable XS
- Defensive copying
- Zero-copy sharing between “normal” agents
  MMU protected page sharing?

Need immutable bulk binary data

(stage 1 problem statement)

- Embedded JS ROM
  tc53 / Moddable XS
- Defensive copying
- Zero-copy sharing between “normal” agents
  MMU protected page sharing?
- OCapN: local rep of bulk binary data
  Like strings, but for bytes

Need immutable bulk binary data

(stage 1 problem statement)

- Embedded JS ROM
  tc53 / Moddable XS
- Defensive copying
- Zero-copy sharing between “normal” agents
  MMU protected page sharing?
- OCapN: local rep of bulk binary data
  Like strings, but for bytes
- Frozen TypedArrays
  Cannot be practically shimmed

Where to start?

- Arrays?
- TypedArrays?
- DataViews?
- Blob?
- Limited ArrayBuffers proposal?
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- TypedArrays?
- DataViews?
- Blob?
- Limited ArrayBuffers proposal?
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- ~~TypedArrays?~~ (but still what we normally want)
- DataViews?
- Blob?
- Limited ArrayBuffers proposal?
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- ~~TypedArrays?~~ (but still what we normally want)
- ~~DataViews?~~ (same problem)
- Blob?
- Limited ArrayBuffers proposal?
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- ~~TypedArrays?~~ (but still what we normally want)
- ~~DataViews?~~ (same problem)
- ~~Blob?~~ (web api with mime type, …)
- Limited ArrayBuffers proposal?
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- ~~TypedArrays?~~ (but still what we normally want)
- ~~DataViews?~~ (same problem)
- ~~Blob?~~ (web api with mime type, …)
- ~~Limited ArrayBuffers proposal?~~ (did get stage 1)
- ArrayBuffers?

Where to start?

- ~~Arrays?~~ (but maybe struct-arrays?)
- ~~TypedArrays?~~ (but still what we normally want)
- ~~DataViews?~~ (same problem)
- ~~Blob?~~ (web api with mime type, …)
- ~~Limited ArrayBuffers proposal?~~ (did get stage 1)
- ArrayBuffers?

```javascript
const consumeIntoNetstring = data => {
  // Transfer to a new ArrayBuffer with room for the netstring framing.
  // https://en.wikipedia.org/wiki/Netstring
  const prefix = new TextEncoder().encode(`${data.length}:`);
  const buf = data.buffer.transfer(prefix.length + data.length + 1);

  // Frame the data.
  const tmpArr = new Uint8Array(buf);
  tmpArr.copyWithin(prefix.length, 0);
  tmpArr.set(prefix);
  tmpArr[tmpArr.length - 1] = 0x2C;

  // Transfer to an immutable ArrayBuffer backing a frozen Uint8Array.
  const frozenNetstring = Object.freeze(new Uint8Array(buf.transferToImmutable()));
  assert(buf.detached);
  return frozenNetstring;
};

const input = new TextEncoder().encode('hello world!');
const result = consumeIntoNetstring(input);
assert(Object.isFrozen(result));
try { result[0] = 0; } catch (_err) {}
try { new Uint8Array(result.buffer)[0] = 1; } catch (_err) {}
try { result.buffer.transferToImmutable(); } catch (_err) {}
assert(String.fromCharCode(...result) === '12:hello world!,');
```

```
Object.freeze(new Uint8Array(buf.transferToImmutable()));
```

Just want frozen `TypedArray`

```
Object.freeze(new Uint8Array(buf.transferToImmutable()));
```

Original ArrayBuffer API

```
slice(start?: number, end?: number) :ArrayBuffer
```

```
get byteLength: number
```

Current ArrayBuffer API

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
resize(len: number) :void
slice(start?: number, end?: number) :ArrayBuffer
```

```
get detached: boolean
get resizable: boolean
get byteLength: number
get maxByteLength: number
```

Resizable ArrayBuffer flavor

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
resize(len: number) :void
slice(start?: number, end?: number) :ArrayBuffer
```

```
get detached: false
get resizable: true
get byteLength: number
get maxByteLength: number
```

Non-Resizable ArrayBuffer flavor

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
~~resize(len: number) :void~~
slice(start?: number, end?: number) :ArrayBuffer



get detached: false
get resizable: false
get byteLength: number
get maxByteLength: same number
```

Detached ArrayBuffer flavor

```
~~transfer(len?: number) :ArrayBuffer~~
~~transferToFixedLength(len?: number) :ArrayBuffer~~
~~resize(len: number) :void~~
~~slice(start?: number, end?: number) :ArrayBuffer~~


get detached: true
~~get resizable: boolean~~
~~get byteLength: number~~
~~get maxByteLength: number~~
```

Proposed ArrayBuffer API

```
transfer(len?: number) :ArrayBuffer
transferToFixedLength(len?: number) :ArrayBuffer
resize(len: number) :void
slice(start?: number, end?: number) :ArrayBuffer
transferToImmutable() :ArrayBuffer
get immutable: boolean
get detached: boolean
get resizable: boolean
get byteLength: number
get maxByteLength: number
```

Immutable ArrayBuffer flavor

```
~~transfer(len?: number) :ArrayBuffer~~
~~transferToFixedLength(len?: number) :ArrayBuffer~~
~~resize(len: number) :void~~
slice(start?: number, end?: number) :ArrayBuffer
~~transferToImmutable() :ArrayBuffer~~
get immutable: true
get detached: false
get resizable: false
get byteLength: number
get maxByteLength: same number
```

10.4.5.1 [[GetOwnProperty]] (*P*)

The [[GetOwnProperty]] internal method of a TypedArray *O* takes argument *P* (a property key) and returns a normal completion containing either a Property Descriptor or **undefined**. It performs the following steps when called:

```
1. If P is a String, then
    a. Let numericIndex be CanonicalNumericIndexString(P).
    b. If numericIndex is not undefined, then
        i. Let value be TypedArrayGetElement(O, numericIndex).
        ii. If value is undefined, return undefined.
        iii. Let mutable be true.
        iv. If IsImmutableBuffer(O.[[ViewedArrayBuffer]]) is true, set mutable to false.
        v. Return the PropertyDescriptor { [[Value]]: value, [[Writable]]: ~~true~~ mutable,
           [[Enumerable]]: true, [[Configurable]]: ~~true~~ mutable }.
2. Return OrdinaryGetOwnProperty(O, P).
```

TypedArray on …

Resizable ArrayBuffer	Cannot* ~~`preventExtensions()`~~
Non-Resizable ArrayBuffer	Can `preventExtensions()`
Detached ArrayBuffer	~~Useless~~
Immutable ArrayBuffer	Can `freeze()`

structuredClone on …

Resizable ArrayBuffer	Copy / `transfer()`
Non-Resizable ArrayBuffer	Copy / `transfer()`
Detached ArrayBuffer	~~Useless / Useless~~
Immutable ArrayBuffer	Zero-copy sharing / ~~No transfer~~

Open Questions

Open Questions

```
transferToImmutable(len?: number) :ArrayBuffer
```

Open Questions

```
transferToImmutable(len?: number) :ArrayBuffer
```

Zero-copy slices?

```
sliceToImmutable(start?: number, end?: number) :ArrayBuffer
```

Open Questions

`transferToImmutable(len?: number) :ArrayBuffer`

Zero-copy slices?

`sliceToImmutable(start?: number, end?: number) :ArrayBuffer`

When/how to report failure to mutate?

Open Questions

`transferToImmutable(len?: number) :ArrayBuffer`

Zero-copy slices?

`sliceToImmutable(start?: number, end?: number) :ArrayBuffer`

When/how to report failure to mutate?

Really orthogonal to `SharedArrayBuffer`?

Status

Status

- Draft spec text (DataView too)
  https://github.com/Agoric/tc39-proposal-immutable-arraybuffer
  https://papers.agoric.com/tc39-proposal-immutable-arraybuffer

Status

- Draft spec text (DataView too)
  https://github.com/Agoric/tc39-proposal-immutable-arraybuffer
  https://papers.agoric.com/tc39-proposal-immutable-arraybuffer

- Partial shim — “secure” but cannot “fix” TypedArray
  https://github.com/endojs/endo/tree/master/packages/immutable-arraybuffer

Status

- Draft spec text (DataView too)
  https://github.com/Agoric/tc39-proposal-immutable-arraybuffer
  https://papers.agoric.com/tc39-proposal-immutable-arraybuffer
- Partial shim — “secure” but cannot “fix” TypedArray
  https://github.com/endojs/endo/tree/master/packages/immutable-arraybuffer
- Stage 1?

Stage 2?

- Wrote spec text to be stage 2 ready
- Partial shim has partial (non-262) tests