

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!!!
- 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 second.

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

`sliceToImmutable(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.

Driven by implementor feedback.

But when in doubt, throw.

Moddable XS implementation throws.

Questions? Stage 2?

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