Transferable Objects: Lightning Fast!

By Eric Bidelman

Engineer @ Google working on web tooling: Headless Chrome, Puppeteer, Lighthouse

Chrome 13 introduced sending `ArrayBuffer`s to/from a Web Worker using an algorithm called structured cloning. This allowed the `postMessage()` API to accept messages that were not just strings, but complex types like `File`, `Blob`, `ArrayBuffer`, and JSON objects. Structured cloning is also supported in later versions of Firefox.

Faster is better

Structured cloning is great, but it's still a copy operation. The overhead of passing a 32MB `ArrayBuffer` to a Worker can be hundreds of milliseconds. New versions of browsers contain a huge performance improvement for message passing, called Transferable Objects.

With transferable objects, data is transferred from one context to another. It is zero-copy, which vastly improves the performance of sending data to a Worker. Think of it as pass-by-reference if you're from the C/C++ world. However, unlike pass-by-reference, the 'version' from the calling context is no longer available once transferred to the new context. For example, when transferring an `ArrayBuffer` from your main app to Worker, the original `ArrayBuffer` is cleared and no longer usable. Its contents are (quiet literally) transferred to the Worker context.

To play with transferables, there's a new version of `postMessage()` that supports transferable objects:

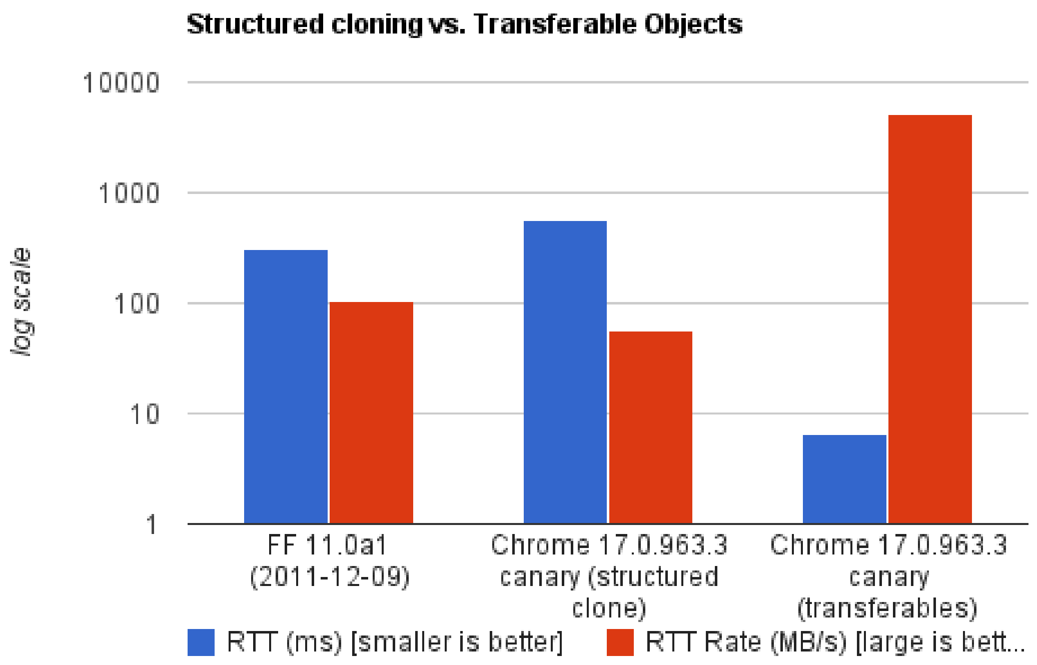
```
worker.postMessage(arrayBuffer, [transferableList]);
window.postMessage(arrayBuffer, targetOrigin, [transferableList]);
```

For the worker case, the first argument is the `ArrayBuffer` message. The second argument is a list of items that should be transferred. In this example, you'd specify the `arrayBuffer` in the transerable list.

Benchmark demo

To see the performance gains of transferrables, I've put together a demo.

The demo sends a 32MB `ArrayBuffer` to a worker and back using `postMessage()`. If your browser doesn't support transferables, the sample falls back to structured cloning. Averaging 5 runs in different browsers, here's what I got:



On a MacBook Pro/10.6.8/2.53 GHz/Intel Core 2 Duo, FF was the fastest using structured cloning. On average, it took 302ms to send the 32MB `ArrayBuffer` to a worker and post it back to the main thread (RRT - Round Trip Time). Comparing that with transferables, the same test took 6.6ms. That is a huge perf boost!

Having these kinds of speeds allows massive WebGL textures/meshes to be seamlessly passed between a Worker and main app.

Feature detecting

Feature detecting is a bit tricky with this one. My recommendation is to send a small `ArrayBuffer` to your worker. If the buffer is transferred and not copied, its `.byteLength` will go to 0:

```
var ab = new ArrayBuffer(1);
worker.postMessage(ab, [ab]);
if (ab.byteLength) {
  alert('Transferables are not supported in your browser!');
} else {
  // Transferables are supported.
}
```

Support: Currently Chrome 17+, Firefox, Opera, Safari, and IE10+

Updated (2011-12-13): Code snippet to show `webkitPostMessage()` signature is different for window and worker. *Updated (2016-11-03):* Removed vendor prefixes and updated code snippets

Last updated July 2, 2018.