Frozen Realms

Mark S. Miller (Google)
erights@google.com

Chip Morningstar (PayPal)
cmorningstar@paypal.com

Caridy Patiño (Salesforce)
cpatino@salesforce.com

Ecma TC39 March 2016

Objective

Enable safe, composable interoperation with code from disparate, uncoordinated originators

- Safe plugins
- Embedded IDEs
- My Yahoo! Modules
- Google Sheets

Objective

Enable safe, <u>composable</u> interoperation with code from disparate, uncoordinated originators

We want to assemble systems out of separately produced pieces

- since we might have lots of pieces, we'd like them to be inexpensive

Objective

Enable safe, composable <u>interoperation</u> with code from disparate, uncoordinated originators

- exchange data with
- provide services to
- consume services from

Objective

Enable safe, composable interoperation with code from disparate, uncoordinated originators

- things don’t clobber other things
- your invarients don’t interfere with my invariants

Or they might just be bad programmers.

Objective

Enable safe, composable interoperation with code from <u>disparate, uncoordinated originators</u>

The various originators don’t necessarily trust each other

- even if they do, they aren’t necessarily coordinated
    - different people make different assumptions
    - different code relies on different invariants

Strategy

Create a single, shared *frozen realm*

- in this realm all primordials are transitively immutable
- its global object (the *frozen-global*) is also transitively immutable

Add a builtin function to provide a new isolated *sub-realm* based on an existing realm

- each has its own (mutable) global that inherits from its super-global
- each has its own `eval` that evaluates in the scope of the sub-realm

Provide access to a sub-realm's `eval`

- so code from outside can be executed inside

Proposed API

The (one) immutable frozen realm

```
Realm.TheFrozenRealm
realm.spawn(endowments)
realm.eval(src)
```

Evaluate `src` inside `realm` using `realm`'s own `eval` function

Create a new sub-realm based on `realm`, with optional `endowments` copied onto its new (mutable) global

Example

Confined computation

```
function confine(src, endowments) {
  return Realm.TheFrozenRealm.spawn(endowments).eval(src);
}
```

Example

Interoperation across realm boundaries

```
function Counter() {
  let count = 0;
  return Object.freeze({
    incr: Object.freeze(() => ++count),
    decr: Object.freeze(() => --count)
  });
}
const counter = new Counter();

// ...obtain billSrc and joanSrc from possibly untrusted clients...
const bill = confine(billSrc, {change: counter.incr});
const joan = confine(joanSrc, {change: counter.decr});
```

Example

Compartments isolated by membranes

```
function makeCompartment(src, endowments) {
  const {wrapper,
         revoke} = makeMembrane(confine);
  return {wrapper: wrapper(src, endowments),
          revoke};
}

// ...obtain billSrc and joanSrc from untrusted clients...
const {wrapper: bill,
       revoke: killBill} = makeCompartment(billSrc, endowments);
const {wrapper: joan,
       revoke: killJoan} = makeCompartment(joanSrc, endowments);

// ... introduce mutually suspicious Bill and Joan to each other...
// ... use both ...
killBill();
// ... Bill is inaccessible to us and to Joan. GC can collect Bill ...
```

Details

A few primordials provide non-determinism

- specifically: `Date.now()`, `new Date()`, `Math.random()`
- these need to be removed or disabled in the frozen realm

Each sub-realm gets its own `eval` and `Function` constructor

- each a fresh object, prototype is its super-realm counterpart
- each evaluates in its own realm's global scope
- a sub-realm is lightweight — 4 objects (realm + global + 2 evaluators)

Frozen realm forbids objects that aren't transitively immutable

- like `window`, `document`, or `XMLHttpRequest`
- don't panic — these can be added back in by endowment when needed
- better yet, attenuated versions of these can be endowed instead

Endowment Example

Putting Date and Math back the way they were

```
function makeColdRealm(GoodDate, goodRandom) {
  const goodNow = GoodDate.now;
  const {Date: SharedDate, Math: SharedMath} = Realm.TheFrozenRealm;
  function FreshDate(...args) {
    if (new.target) {
      if (args.length === 0) {
        args = [+goodNow()];
      }
      return Reflect.construct(SharedDate, args, new.target);
    } else {
      return String(GoodDate());
    }
  }
  FreshDate.__proto__ = SharedDate;
  FreshDate.now = Object.freeze(() => +goodNow());
  FreshDate.prototype = SharedDate.prototype;   // so instanceof works
  FreshDate.name = SharedDate.name;

  const FreshMath = {
    __proto__: SharedMath,
    random() { return +goodRandom(); }
  };
  Object.freeze(FreshMath.random);

  const freshRealm = Realm.TheFrozenGlobal.spawn({
    Date: Object.freeze(FreshDate),
    Math: Object.freeze(FreshMath)
  });
  Object.freeze(freshRealm.global);
  Object.freeze(freshRealm.global.eval);
  Object.freeze(freshRealm.global.Function);
  return freshRealm;
}
```

Open Questions

Relationship to CSP?

- should CSP "no script evaluation" settings exempt the frozen realm's evaluators?
- or should CSP be extended to express differential prohibition?

Best way to cope with the override mistake?

- OMG this is so horrible

Best way to censor the Date() constructor?

- we said: throw `TypeError`, but it's a somewhat arbitrary pick

Name bikeshedding

- we could go on at length and no doubt will...

Links

Draft spec document

https://github.com/FUDCo/frozen-realms

Relevant formal semantics

http://research.google.com/pubs/pub37199.html

Google SES shim — implementing this the slow, hard way

https://github.com/google/caja/tree/master/src/com/google/caja/ses