Realms Update

Dave, Caridy, Mark, Yehuda

Realms Recap

- Low-level, "extensible web" API
- Target for polyfills and transpilers
- Unifies iframes and node's "vm" module

Goals Today

- Present recent progress on realms API
- Show viability of userland registries and loaders
- Gather input for work towards Stage 2

Example: a tiny, fixed registry

```
class TinyLittleRealm extends Realm {

  constructor() {
    super();
    this.#registry = new TinyLittleRegistry(this, jquery);
  }

  [Realm.import](name, referer) {
    if (["a", "b"].indexOf(name) < 0) {
      throw new ReferenceError("unrecognized module: " + name);
    }
    return this.#registry.graph().then(graph => graph[name]);
  }
}

let realm = new TinyLittleRealm();

realm.eval("import('a')").then(m => m.a());
```

```
class TinyLittleRegistry {

  constructor(realm, jquery) {
    this.#realm = realm;
    this.#jquery = jquery;
  }

  @memoize records() {
    return Promise.all([fetch("a.js"), fetch("b.js")])
                  .then(sources => sources.map(s => this.#realm.parseModule(s)));
  }

  // ...
}
```

```
class TinyLittleRegistry {

  // ...

  @memoize records() {
    return Promise.all([fetch("a.js"), fetch("b.js")])
                  .then(sources => sources.map(s => this.#realm.parseModule(s)));
  }

  @memoize async graph() {
    let { a, b } = await this.records();

    a.add("./b.js", b);
    a.add("jquery", this.#jquery);

    b.add("./a.js", a);
    b.add("jquery", this.#jquery);

    a.link();
    b.link();

    return { a, b };
  }

}
```

Concepts

- realm.parseModule(src)
- realm[Realm.import](specifier, referer)
- module.add(requestedName, otherModule)
- module.link()

Example: a dynamic loader

```
class TinyLittleRealm extends Realm {

  constructor() {
    super();
    this.#loader = new TinyLittleLoader(this, jquery);
  }

  async [Realm.import](name, referrer) {
    let module = await this.#loader.load(name).linked;
    module.ensureEvaluated();
    return module;
 }
}

let realm = new TinyLittleRealm();

realm.eval("import('a')").then(m => m.a());
```

```
class TinyLittleLoader {

  constructor(realm, jquery) {
    this.#realm = realm;
    this.#jquery = jquery;
    this.#registry = new Dict();
  }

  load(specifier) {
    if (specifier === 'jquery') {
      return this.#jquery;
    }

    let name = specifier.replace(/^\.\//, "")
                        .replace(/\.js$/, "");

    if (!this.#registry.has(name)) {
      this.register(name);
    }

    return this.#registry.get(name);
  }

  // ...

}
```

```
class TinyLittleLoader {

  // ...

  register(name) {
    let module = fetch(specifier).then(source => this.#realm.parseModule(source));

    let complete = module.then(module => {
      let specifiers = module.requestedNames();
      return Promise.all(specifiers.map(specifier => {
        let entry = this.load(specifier);
        return entry.complete.then(() => entry.module.then(dependency => {
          module.add(specifier, dependency);
          return dependency;
        }));
      }));
    });

    let linked = module.then(module => complete.then(() => {
      module.link();
      return module;
    }));

    this.#registry.set(name, { module, complete, linked });
  }

}
```

Concepts

- module.ensureEvaluated()

- module.requestedNames()

Takeaways

- Userland registries and loaders should be possible
- Dramatically reduce the API surface for hooks
- We have a draft spec and polyfill; more work to do