`new Global`

status update

Zbyszek Tenerowicz (ZTZ) @naugtur.pl

Motivation

- Domain Specific Languages
- Test runners
- Principle of Least Authority
- Isolation of unreliable code

Example - DSL

```
const dslGlobal = const new Global();
dslGlobal.describe = () => {};
dslGlobal.before = () => {};
dslGlobal.after = () => {};

const source = await import.source(entrypoint);
await dslGlobal.eval('s => import(s)')(source);
```

```
dslGlobal.document = mockDomApi;
```

new Global

```
interface Global {
  constructor({
    keys?: string[],
    importHook?: ImportHook,
    importMeta?: Object,
  })

  Global: typeof Global,
  eval: typeof eval,
  Function: typeof Function,
  // and all permutations of Async and Generator

  // and properties copied from globalThis filtered by keys
}
```

Conceptual changes

- replaces the `Evaluators` proposal
- avoids adding new entities
- host creates the global object
- not opinionated on minimal set of globals

> “ `Global` picks up from the previous proposal for `Evaluators` and results from an observation that an object conveniently containing all evaluators alreaedy exists in the spec and all we need to do is expose a constructor for it.
> It also eliminates the concern where evaluators accepting any globalThis to use would clash with the host implementation's desire to use special objects only the host can create.
> No API to set a custom reference as global context for evaluators if it's not created via `new Global`
> When a new global is created it inherits all properties from parent global unless user specifies a list. Spec offers no opinions on minimal global, only demands that all evaluators are present. ”

Details

- allows mutating `(new Global()).globalThis` before evaluation
- by default copy all properties from `globalThis`
- properties: `Global` and all evaluators have their internal slots relating them to the new *global*

Details - invariants

```
globalThis.x = {};
const thatGlobal = new globalThis.Global({
  keys: Object.keys(globalThis),
});
thatGlobal.eval !== thisGlobal.eval;
thatGlobal.Global !== thisGlobal.Global;
thatGlobal.Function !== thisGlobal.Function;
thatGlobal.eval("Object.getPrototypeOf(async () => {})") !==
  Object.getPrototypeOf(async () => {});
thatGlobal.eval("Object.getPrototypeOf(function *() {})") !==
  Object.getPrototypeOf(function* () {});
thatGlobal.eval("Object.getPrototypeOf(async function *() {})") !==
  Object.getPrototypeOf(async function* () {});
const source = new ModuleSource("export default {}");
(await thatGlobal.eval("(...args) => import(...args)")(source)) !==
  (await import(source));
thatGlobal.ModuleSource === thisGlobal.ModuleSource;
thatGlobal.Array === thisGlobal.Array;
thatGlobal.x === thisGlobal.x;
```

Overlap with Module Harmony

```
const globalThat = new Global({
  importHook(specifier) {
    log(`global ${specifier}`);
    return new ModuleSource("");
  },
});
const source = new globalThat.ModuleSource(
  `
  import 'static-import';                                    // local static-import
  eval('import("direct-eval-import")');                      // local direct-eval-import
  globalThis.eval('import("indirect-eval-import")');  // global indirect-eval-import
  new Function('return import("function-import")');   // global function-import
`,
  {
    importHook(specifier) {
      log(`local ${specifier}`);
    },
  }
);
await import(source);
```