Ecma TC39
November 18, 2014

ES6 Editor’s Status and Issues

Allen Wirfs-Brock
Rev28 Draft

• Modules
  – Removed loader pipeline and Reflect.Loader API (functionality being transferred to separate specification)
  – Stream-lined module linking semantics for declarative modules.
  – Removed Module declaration
  – Update Import declaration to include module imports.
  – Updated default export syntax and semantics to support export of anonymous default functions
  – Added Module Environment Records and indirect (import) bindings
  – Added Module evaluation jobs
  – Added Host hooks for module name normalization and source access.

• In Rev29, name normalization fixed to support relative naming
Rev28 Draft

• Interim Subclass Instantiation Reform
  – Changed ordinary object creation to dispatch object allocation through `[[CreateAction]]` internal slot instead of `@@create` method.
  – Converted all `@@create` methods into `CreateAction` abstract operations.
  – Eliminated `Symbol.create` and `@@create`.
  – `super` without an immediately following property specifier is now illegal in all `MethodDefinition` (no more implicit `super` using current method name)
  – `super` in a constructor call expression references the constructor’s `[[Prototype]]`
  – `Function.prototype.toMethod` no longer takes an optional name argument
Rev28 Draft

• Finished up ES6 eval function semantics
• Eliminated unused abstract operations, PromiseAll, PromiseCatch, PromiseThen
• Modified Promise.all so specification internally uses a List instead of an Array to accumulate result promises
• Added @@iterator property to %IteratorPrototype%
• Added requirement that the object returned by ordinary object [[Enumerate]] must inherit from %IteratorPrototype%
• Removed own @@iterator properties from various standard iterators, they now inherit it from %IteratorPrototype%
• Updated ToPropertyKey to accept Symbol wrapper objects, similar to how other primitive coercion abstract operations handle wrapper objects
• ToNumber now recognizes binary and octal string numeric values.
• Significant fix to destructuring assignment where the rest assignment target is itself a destructuring pattern
• Updated Annex A Grammars to match ES6
End Game Planning

• Needed:
  – One paragraph summary of ES6 goals for Introduction
  – Clause 4 – Language Overview. Needs to be updated to reflect ES6 features

• Readers, Readers, Readers ...

• Ecma-402 Edition 2, review.

• How will we resolve last minute issues?
Assignment to a const: Static Error?

- https://esdiscuss.org/topic/throwing-errors-on-mutating-immutable-bindings
  
  https://bugs.ecmascript.org/show_bug.cgi?id=3253

  const x = 42;
  x=32; //early error???

- es-discuss consensus: eliminate early error
- However, current spec. draft (legacy) ES5 semantics only throws on assignment to an immutable binding in strict mode:

  “don’t use strict”;
  Object.defineProperty(this, ‘globalReadOnly’, {value: ‘readonly’});
  var func = function f() {
    f = undefined; //silently skips assignment
    undefined = 42; //silently skips assignment
    Infinity = 0;  //silently skips assignment
    globalReadOnly = 0; //silently skips assignment
  };
  func(); //no exception thrown

- Should assignment to const also be silent in non-strict mode? Exception will require some new spec. mechanisms.
MooTools conflict with String.prototype.contains

- [https://bugzilla.mozilla.org/show_bug.cgi?id=1075059](https://bugzilla.mozilla.org/show_bug.cgi?id=1075059)

Options:
- Leave as is, break some web sites
- Remove ‘contains’ method
- Rename ‘contains’ to something else. What?
- Rename to “includes”

Also, Outlook web client issue with Array.prototype.values
- [https://esdiscuss.org/topic/array-prototype-values-is-not-web-compat-even-with-unscopables](https://esdiscuss.org/topic/array-prototype-values-is-not-web-compat-even-with-unscopables)

How should be deal with similar issues as we approach ES6 ship date??
Global let shadowing of non-configurable global properties

- [https://esdiscuss.org/topic/late-shadowing-of-globals-esp-undefined](https://esdiscuss.org/topic/late-shadowing-of-globals-esp-undefined)
- [https://bugs.ecmascript.org/show_bug.cgi?id=3301](https://bugs.ecmascript.org/show_bug.cgi?id=3301)
- For example: let undefined = 666
- Issues
  - When are/aren’t global lets allowed to shadow an already existing property of a global object
  - Are built-in globals equivalent to global vars or are they just properties of global object
  - Make it illegal to shadow a global property would mean future global properties are breaking changes
- Proposal: Runtime error when instantiating a script if a lexical declaration shadows a non-configurable own property of global object
Zepto broken by new this.constructor pattern in some Array methods

• [https://bugs.ecmascript.org/show_bug.cgi?id=3256](https://bugs.ecmascript.org/show_bug.cgi?id=3256)

• Intended to produce same subclass as original this value.

• But Zepto does:

```javascript
var obj = [1,2,3];
obj.__proto__ = { slice: Array.prototype.slice };
var res = obj.slice(2);
Array.isArray(res); // true in ES5, false in ES6.
```

• this.constructor is Object!
4. If $O$ is an exotic Array object, then
   a. Let $C$ be $\text{Get}(O, "\text{constructor}").$
   b. ReturnIfAbrupt($C$).
   c. If $\text{IsConstructor}(C)$ is $\text{true}$, then
      i. Let $this\text{Realm}$ be the running execution context’s Realm.
      ii. If $\text{SameValue}(this\text{Realm}, \text{GetFunctionRealm}(C))$ is $\text{true}$, then
         1. Let $A$ be the result of calling the $\text{[[Construct]]}$ internal method of $C$ with argument $(0)$.

5. If $A$ is $\text{undefined}$, then
   a. Let $A$ be $\text{ArrayCreate}(0)$. 
Zepto Proposed Fix

4. Let \( C \) be Get\((O, "constructor")\).
5. ReturnIfAbrupt\((C)\)
6. If IsConstructor\((C)\) is \textbf{true}, then
   a. Let \textit{thisRealm} be the running execution context’s Realm.
   b. If SameValue\((\textit{thisRealm}, \text{GetFunctionRealm}(C))\) is \textbf{true}, then
      i. Let \textit{species} be Get\((C, @@\text{species})\);
      ii. ReturnIfAbrupt\((\textit{species})\)
      iii. If IsConstructor\((\textit{species})\) is \textbf{true}, then
         1. Let \( A \) be the result of calling the \([[\text{Construct}]]\) internal method of \textit{species} with argument \((0)\).
7. If \( A \) is \textbf{undefined}, then let \( A \) be ArrayCreate\((0)\).

\ @@\text{species} \Rightarrow \ @@\text{copyConstructor} ??
Template String call site caching and eval

- [https://bugs.ecmascript.org/show_bug.cgi?id=3305](https://bugs.ecmascript.org/show_bug.cgi?id=3305)

```javascript
let world = "world";
let t = "tag`hello, ${world}.`";
eval(t);
eval(t);
eval(t);
new Function(t)();
new Function(t)();
tag`hello, ${world}.`;
```

- How many call sites? 5, 3, 2, or 1?

- Meeting decision: 1
Array.isArray

Array[Symbol.isArray]] = true;
Array.isArray = function (obj) {
  let constructor = obj.constructor;
  If (typeof constructor != 'function') return false;
  let isArrayC =
    Object.getOwnPropertyDescriptor(constructor,"isArray");
  if (isArrayC) {
    If (isOrdinary(obj) return false;
      //if (isProxy(obj)) return isArrayC.value(proxyTarget(obj));
  }
  return !!constructor[Symbol.isArray];
}
Also

• Change Array.prototype.concat to do the new Array.isArray test instead of using @@isConcatSpreadable
  — If Array.isArray(obj) is true, concat will flatten the object

• Change JSON.stringify to Array.isArray test where it currently checks for an exotic array object.
  — If Array.isArray(obj) is true, stringify obj will use [ ] notation

• Give %TypedArray% a true valued @@isArray property
  — Array.isArray(new Int32Array(10)) will be true
  — Verify that if doesn’t break anything