

Object Instantiation Redo

<https://gist.github.com/allenwb>

<https://gist.github.com/allenwb/291035fbf910eab8e9a6>

<https://gist.github.com/allenwb/53927e46b31564168a1d>

Contributors

- Allen
 - Arv
 - Mark
 - Dmitry
-
- + es-discuss and private feedback

Main Issues

- @@create can expose uninitialized instances of built-in and host objects
- Necessitates numerous dynamic “is it initialized” checks in order to guarantee the invariants of such objects

Original Ideal From Claude Pauche

```
class C extends B {  
  constructor(...args) {  
    /* 1: preliminary code that doesn't contain calls to a super-method */  
    /* this in TDZ */  
    /* 2: call to a super-constructor */ super(...whatever);  
    /* this defined */  
    /* 3: the rest of the code */  
  }  
}
```

- Added “receiver” argument to `[[Construct]]` that passes the constructor that `new` was originally applied to.

Additional Idea Presented at Last Meeting

- `new*` token
- Value is the “receiver” parameter from `[[Construct]]` or undefined if `[[Call]]`
- Can be used to discriminate “called as constructor” and “called as function”
- Provides access to original constructor for object initialization/initialization
 - `Object.create(new*.prototype);`

`new*` has been replaced by `new^`

Evolved Design

new super()

- Use `new super()` rather than `super()` to “invoke superclass’ constructor”
 - `new super();` is always a `[[Construct]]` invocation
 - `super();` is always a `[[Call]]` invocation
- Didn’t want to further confuse “called as a constructor” and “called as a function”.
 - `<id>()` -- always means “called as function”
 - `new <id>()` – always means “called as constructor”
 - Even when `<id>` is `super`

this = new super()

- Original proposal had `this` in TDZ until explicit `super()` call. (now `new super()`)
 - Invisibly assigned to `this`
- Update proposal eliminates the implicit assignment by `new super()`.
- But allows an explicit assignment to `this`
 - Only in constructors
 - Only a single dynamic assignment
 - Subsequent assignments throw ReferenceError

this = <expr>

- RHS of `this` assignment in a constructor isn't limited to `new super()`
- May be any object valued expression:
this = new super();
this = {x:1, y:2};
this = Object.setPrototypeOf([], new^.prototype);
this = new Proxy(new super(), handler);

Works in both class constructors and function constructors

```
SubArray.__proto__=Array;  
SubArray.prototype=Object.create([].prototype);  
function SubArray(...args) {  
    if (!this^) this = new SubArray(...args);  
    else this = new super(...args);  
}
```

Default object allocation (Base Classes)

- Class constructors without an `extends` and basic (function) constructors...
- ... Assign an new ordinary object to `this` if body does not have an explicit `this=`.
- These continue to mean the same thing:
 - ```
class Base {
 constructor(x) {
 this.x = x;
 }
}
```
  - ```
function Base(x) {  
    this.x = x;  
}
```

Unqualified super references

- Until now ES6 has said that `super()` means the same thing as `super.<method name>()`
 - Implicit property access
 - Requires setup using `toMethod` (implicit or explicit)
- `super` in constructor needs to mean “this constructor’s `[[Prototype]]`”, not `[[HomeObject]].prototype.constructor`
- It would be confusing if `super()` means something completely different in a constructor from what it means in a non-constructor method
- Is this going to be used as a constructor or a method?

```
function f() {return super();}
```

Eliminate unqualified `super` reference in non-constructor methods

```
class C {  
    foo() {  
        //return super();    // now syntax error  
        return super.foo(); //must say this instead  
    }  
}
```

- Unqualified `super` only allowed in class constructors and function definitions
- Regular methods must explicitly qualify `super` references with a property access

Default Value of `this` in derived constructors that don't assign to `this`

- Some alternatives:
 - `this = new super();` //super new with no arguments
 - `this = new super(...arguments);` //super new all args
 - `this = Object.create(new^.prototype);` // ordinary obj
 - No value, `this` in TDZ at constructor start
- Most controversial part of design discussion

The winner: no default `this` in derived constructors

- Eliminates issues of what arguments` to pass to implicit `new super()`
- Must assign to `this` in derived constructor before referencing it.