Declaring static constants in ES6 classes?

Asked 4 years ago Active 1 month ago Viewed 201k times

268

46

I want to implement constants in a `class`, because that's where it makes sense to locate them in the code.

So far, I have been implementing the following workaround with static methods:

```
class MyClass {
    static constant1() { return 33; }
    static constant2() { return 2; }
    // ...
}
```

I know there is a possibility to fiddle with prototypes, but many recommend against this.

Is there a better way to implement constants in ES6 classes?

javascript class constants ecmascript-6

edited Sep 18 '15 at 16:59
sdgluck
11.8k 1 34 56

asked Sep 18 '15 at 8:22
Jérôme Verstrynge
30.1k 71 234 401

4 Personally I just use uppercase VARNAMES, and tell myself to not touch them ;) – twicejr Jul 29 '16 at 12:18

2 @twicejr I think this is not the same, for static variables can be accessed without first instantiating an object of that class? – Lucas Morgan Feb 7 '17 at 20:18

9 Answers

Here's a few things you could do:

```
export const constant1 = 33;
```

And import that from the module where necessary. Or, building on your static method idea, you could declare a `static` get accessor:

```
const constant1 = 33,
      constant2 = 2;
class Example {

  static get constant1() {
    return constant1;
  }

  static get constant2() {
    return constant2;
  }
}
```

That way, you won't need parenthesis:

```
const one = Example.constant1;
```

Babel REPL Example

Then, as you say, since a `class` is just syntactic sugar for a function you can just add a non-writable property like so:

```
class Example {
}
Object.defineProperty(Example, 'constant1', {
    value: 33,
    writable : false,
    enumerable : true,
    configurable : false
});
Example.constant1; // 33
Example.constant1 = 15; // TypeError
```

It may be nice if we could just do something like:

But unfortunately this class property syntax is only in an ES7 proposal, and even then it won't allow for adding `const` to the property.

edited Sep 18 '15 at 8:51

answered Sep 18 '15 at 8:41

CodingIntrigue

51.9k 17 137 154

is there any confirmation that static properties get computed once for things like this, or is it safer to use IIFE and add the property manually in the IIFE to avoid repeated construction of return values. I'm worried that if the result of the getter is really heavy, like a 100000 entry JSObject, then the poor getter will have to construct it each time the getter is called. Its easy to test by performance.now/date diff, but it might be implemented differently, its certainly easier to implement getters as literal evaluation rather than advanced decisions whether its constant or not. – Dmitry Aug 16 '17 at 13:35

2 while the above cleverly adds a constant property to a class, the actual value for the constant is "outside" the class definition "{}", which really violates one of the definitions of encapsulation. I guess it is sufficient to just define a constant property "inside" the class and there is no need for the get in this case. – NoChance Oct 5 '17 at 8:43

1 @NoChance Good points. That was just illustrative. There's no reason the getter method couldn't fully encapsulate the value if required. – CodingIntrigue Oct 5 '17 at 8:47

Looking forward to use the ES7 proposal because it looks to me more natural and equivalent to the the majority of OO languages. – Sangimed Apr 13 '18 at 8:50

What it I want to declare constant an instance variable? Can I do something like `this.defineProperty(this, 'constant1', {...})` – Francesco Boi Jul 24 '18 at 12:38

23

I'm using `babel` and the following syntax is working for me:

```
class MyClass {
    static constant1 = 33;
    static constant2 = {
       case1: 1,
       case2: 2,
    };
    // ...
}

MyClass.constant1 === 33
MyClass.constant2.case1 === 1
```

```
npm install --save-dev babel-preset-stage-0

// in .babelrc
{
    "presets": ["stage-0"]
}
```

Update:

currently use `stage-3`

edited Aug 22 at 22:50

answered Aug 24 '16 at 6:09

borracciaBlu

2,654 3 20 34

16 Problem is that constant is reassignable. Op doesn't want that – CodingIntrigue Aug 27 '16 at 20:29

3 FYI, this is now in babel `stage-2` – bmaupin Aug 6 '17 at 19:48

3 those aren't constants – Dave L. Aug 15 '17 at 19:09

1 @CodingIntrigue Would calling `Object.freeze()` on the class fix that? – Antimony Sep 12 '17 at 2:07

1 @Antimony I haven't tested that but I would think so. The problem is it would apply to all properties of the class. Non-static too. – CodingIntrigue Sep 12 '17 at 5:51

22

```
class Whatever {
    static get MyConst() { return 10; }
}

let a = Whatever.MyConst;
```

Seems to work for me.

answered May 10 '18 at 10:30

Benny Jobigan

3,619 1 24 37

1 @PirateApp you can access it anywhere as a static method, even from inside an instance of the class. However, since it's static you can't use `this.MyConst` from inside a `Whatever` instance, you always have to write it like this: `Whatever.MyConst` – TheDarkIn1978 Apr 28 at 16:55

13

In this document it states:

> There is (intentionally) no direct declarative way to define either prototype data properties (other than methods) class properties, or instance property

This means that it is intentionally like this.

Maybe you can define a variable in the constructor?

```
constructor(){
    this.key = value
}
```

edited Sep 18 '15 at 11:08
sdgluck
11.8k 1 34 56

answered Sep 18 '15 at 8:33
DevAlien
2,092 9 15

2 Yes, this can work. Also, I want to mention, that constructor invokes when instance created and for each instance this.key will be not the same. Static method and properties allow us to use them directly from class, without creating instance. There are good and weak points of static methods / properties. – Kirill Gusyatin Feb 23 '17 at 14:08

Constants should be immutable. Assigning to properties on the object during construction will yield properties that can be modified. – philraj Jun 13 '18 at 19:10

10

It is also possible to use `Object.freeze` on you class(es6)/constructor function(es5) object to make it immutable:

```
class MyConstants {}
MyConstants.staticValue = 3;
MyConstants.staticMethod = function() {
```

```
// after the freeze, any attempts of altering the MyConstants class will have no result
// (either trying to alter, add or delete a property)
MyConstants.staticValue === 3; // true
MyConstants.staticValue = 55; // will have no effect
MyConstants.staticValue === 3; // true

MyConstants.otherStaticValue = "other" // will have no effect
MyConstants.otherStaticValue === undefined // true

delete MyConstants.staticMethod // false
typeof(MyConstants.staticMethod) === "function" // true
```

Trying to alter the class will give you a soft-fail (won't throw any errors, it will simply have no effect).

answered May 16 '16 at 19:11

rodrigo.botti
218 1 4 9

3 That soft-fail is pretty scary for those of us coming from other languages - just adapting to the idea that the tools don't help us much in finding errors, now even the runtime won't help. (Otherwise I like your solution.) – Tom Aug 2 '16 at 21:01

I love `Object.freeze()` for enforcing immutability, and have been using it a lot lately. Just don't forget to apply it recursively! – jeffwtribble Mar 6 '17 at 21:47

5

Maybe just put all your constants in a frozen object?

```
class MyClass {

    constructor() {
        this.constants = Object.freeze({
            constant1: 33,
            constant2: 2,
        });
    }

    static get constant1() {
        return this.constants.constant1;
    }
```

```
        //...
    }
}
```

answered Sep 5 '18 at 10:34

aRIEL
71 1 3

The static function can't use the variable 'this'. – PokerFace Jul 24 at 9:44

4

Like https://stackoverflow.com/users/2784136/rodrigo-botti said, I think you're looking for `Object.freeze()` . Here's an example of a class with immutable statics:

```
class User {
  constructor(username, age) {
    if (age < User.minimumAge) {
      throw new Error('You are too young to be here!');
    }
    this.username = username;
    this.age = age;
    this.state = 'active';
  }
}

User.minimumAge = 16;
User.validStates = ['active', 'inactive', 'archived'];

deepFreeze(User);

function deepFreeze(value) {
  if (typeof value === 'object' && value !== null) {
    Object.freeze(value);
    Object.getOwnPropertyNames(value).forEach(property => {
      deepFreeze(value[property]);
    });
  }
  return value;
}
```


1 1


941 1 6 4

1

Here is one more way you can do

```
/*
one more way of declaring constants in a class,
Note - the constants have to be declared after the class is defined
*/
class Auto{
   //other methods
}
Auto.CONSTANT1 = "const1";
Auto.CONSTANT2 = "const2";

console.log(Auto.CONSTANT1)
console.log(Auto.CONSTANT2);
```

Run code snippet Expand snippet

Note - the Order is important, you cannot have the constants above

Usage console.log(Auto.CONSTANT1);

answered Sep 1 '17 at 18:49


user3871424
153 2 4

5 They aren't immutable though – John Harding Sep 14 '17 at 8:31

1

You can create a way to define static constants on a class using an odd feature of ES6 classes. Since statics are inherited by their subclasses, you can do the following:

```
    Object.defineProperty(ConstClass, key, {
      value: map[key],
      writable : false,
      enumerable : true,
      configurable : false
    });
  });
  return ConstClass;
};

class MyClass extends withConsts({ MY_CONST: 'this is defined' }) {
  foo() {
    console.log(MyClass.MY_CONST);
  }
}
```

answered Apr 23 '18 at 18:16

TbWill4321

7,557 2 20 22