A freestanding JavaScript function that uses this is easily mistaken for a constructor

devblogs.microsoft.com/oldnewthing/20230907-00

September 7, 2023

Raymond Chen

As a general rule in JavaScript, when you define a function with the `function` keyword (not with the arrow syntax), the `this` keyword is bound to the invoking object if the function was called as the result of the `.` (dot) operator. Otherwise, it is bound to `undefined` (in strict mode) or the global object (in non-strict mode).

Being bound to `undefined` or the global object is not particularly interesting. In practice, a function uses `this` if it was intended to be called via the `.` (dot) operator, which means in practice that it was defined either by adding it to a prototype or by using the syntactic sugar of an ES6 class declaration.

The only common case where a freestanding function (not stored on a prototype) uses `this` is when it is a constructor. In that case, the `this` keyword is bound to the object being constructed.

Visual Studio Code has a static analyzer that looks for functions that look like constructors. When it finds one, it suggests, “This constructor function may be converted to a class declaration.” I don’t know the details of this static analyzer,[1] but from observation, it appears that the logic for this rule is very simple: A function declared with the `function` keyword that writes to properties of `this` is probably a constructor. (Because there’s rarely any reason for a non-constructor to use `this`.)

This rule generally works, except when it doesn’t.

You can override the normal assignment of `this` by using methods like `call`, `apply`, or `bind`. The `call` and `apply` methods override the value of `this` for a single call of that function. The `bind` method returns a new function that calls the original function with the `this` overridden by the value you specify at bind time.

Visual Studio Code’s static analyzer doesn’t know about these weirdos.

But wait, there’s more.

Some callers intentionally bind `this` to something strange when calling a callback function. For example, the Array iterative methods take an optional second parameter traditionally named `thisArg`. Before calling the callback function, the iterative method binds `this` to the value you passed as the `thisArg`, and your callback function can then use `this` to refer to that object.[2] Another example is HTML DOM event handlers, which bind `this` to the element that generated the event.

If you use either of these features, then the suggestion will misdetect your function as a constructor candidate.

If you don't want to disable the suggestion outright, it may be possible to make small adjustments to your code to avoid the misdetection.

If your function is an event handler, then you can use `e.currentTarget` instead of `this` to access the element that generated the event. This is probably more idiomatic anyway, because `this` is easily lost when you do code factoring.

```
// original version

function focusHandler(e) {
    var item = findItemFromElement(this);
    if (item.ready) {
        this.className = "ready";
    } else {
        this.className = "waiting";
    }
}

// refactored, bug introduced

function focusHandler(e) {
    var item = findItemFromElement(this);
    updateClass(item);
}

function updateClass(item) {
    if (item.ready) {
        // Oops: "this" means something different here
        this.className = "ready";
    } else {
        this.className = "waiting";
    }
}

// alternate original version

function focusHandler(e) {
    var item = findItemFromElement(this);
    if (item.ready) {
        e.currentTarget.className = "ready";
    } else {
        e.currentTarget.className = "waiting";
    }
}

// improper refactoring detected

function focusHandler(e) {
    var item = findItemFromElement(this);
    updateClass(item);
}

function updateClass(item) {
    if (item.ready) {
        // error: "e is not defined"
        e.currentTarget.className = "ready";
    } else {
        e.currentTarget.className = "waiting";
```

```
    }
}
```

The "`e` is not defined" error clues you in that the code you factored out also needed `e`, and you forgot to add it as a parameter.

For the Array iterative methods, custom binding of `this` is one of those obscure features that you show off at parties, but rarely something you use in real code. If you really do want to pass a bonus parameter to the callback function, you can bind it explicitly.

```
// original version, false positive
function accumulateTotal(e) {
    this.total += e.value;
}

class Something
{
    updateTotal(items) {
        this.total = 0;
        items.forEach(accumulateTotal, this);
        return this.total;
    }
}

// revised version, avoids false positive
function accumulateTotal(o, e) {
    o.total += e.value;
}

class Something
{
    updateTotal(items) {
        this.total = 0;
        items.forEach(accumulateTotal.bind(null, this));
        return this.total;
    }
}

// revised version, avoids false positive, less obscure

class Something
{
    updateTotal(items) {
        this.total = 0;
        items.forEach(e => accumulateTotal(this, e));
        return this.total;
    }
}
```

[1] You can find the code that implements the "can be converted to a class constructor" suggestion in the function `canBeConvertedToClass` in `suggestionDiagnostics.ts`.

[2] Offer not valid for functions which have already been bound to a `this`, either explicitly via `bind` or implicitly by virtue of being an arrow function.