A Drip of JavaScript

The Uses of 'in' vs 'hasOwnProperty'

Originally published in the A Drip of JavaScript newsletter.

Last issue I briefly mentioned JavaScript's `in` operator, but didn't go into much detail about its use. That's the subject we'll be tackling first today.

The `in` operator will tell you whether an object (or array) has a property name which matches a given string.

```
var fantasyLit = {
    tolkien: "The Lord of the Rings",
    lewis: "The Chronicles of Narnia"
};

// Outputs: true
console.log("tolkien" in fantasyLit);

// Outputs: false
console.log("asimov" in fantasyLit);
```

Looks simple enough, right? But consider this:

```
// Outputs: true
console.log("constructor" in fantasyLit);
```

What's going on here? It turns out that the `in` operator doesn't distinguish between properties created specifically on an object and properties that the object inherited from the prototype chain. In this case `in` is seeing the `constructor` property of `Object.prototype` which all objects inherit from.

It will also return `true` for user-defined prototype properties. For instance:

```
function litList () {}

litList.prototype.addToList = function (author, work) {
    this[author] = work;
};

var fantasyLit = new litList();

fantasyLit.addToList("tolkien", "The Lord of the Rings");

// Outputs: true
console.log("tolkien" in fantasyLit);

// Outputs: false
console.log("asimov" in fantasyLit);

// Outputs: true
console.log("addToList" in fantasyLit);
```

Because of this, using `in` to detect whether an object possesses a given property can be a bit deceptive. Usually we only want to check for properties that belong to the object itself, not its prototype chain. Fortunately, JavaScript has a solution for that. It is called `hasOwnProperty`.

It is a method on `Object.prototype`, which means it is available to all JavaScript objects.

Here is how you use it:

```
function litList () {}

litList.prototype.addToList = function (author, work) {
    this[author] = work;
};

var fantasyLit = new litList();

fantasyLit.addToList("tolkien", "The Lord of the Rings");

// Outputs: true
console.log(fantasyLit.hasOwnProperty("tolkien"));

// Outputs: false
console.log(fantasyLit.hasOwnProperty("asimov"));

// Outputs: false
console.log(fantasyLit.hasOwnProperty("addToList"));
```

Because in JavaScript arrays also inherit from `Object`, they can use `hasOwnProperty` as well, though it is often less useful.

```
var summerMovies = [
    "Iron Man 3",
    "Star Trek: Into Darkness",
    "Man of Steel"
];

// Outputs: true
summerMovies.hasOwnProperty("2");
```

It's important to keep in mind the limits of both `in` and `hasOwnProperty`. While they can tell you that a given property has been declared, they can't tell you whether the property has a "real value".

Consider these examples:

```
// Puts a "declared" property on the global object
// (window in browsers)
var declared;

// Outputs: true
console.log("declared" in window);

// Outputs: true
console.log(window.hasOwnProperty("declared"));

// Outputs: undefined
console.log(declared);

var obj = { myUndefined: undefined };

// Outputs: true
console.log("myUndefined" in obj);

// Outputs: true
console.log(obj.hasOwnProperty("myUndefined"));

// Outputs: undefined
console.log(obj.myUndefined);
```

Another limit that you may encounter is that the `hasOwnProperty` method can be rendered useless if an object happens to define a property named `hasOwnProperty`. For instance:

```
var voldemort = {
    hasOwnProperty: function () { return true; }
};

// Outputs: true
console.log(voldemort.hasOwnProperty("ridikulus"));
```

Because `voldemort` defines its own `hasOwnProperty`, the call never makes it to `Object.prototype.hasOwnProperty`. It's unlikely that you'll run into an object as maliciously constructed as `voldemort`, but it's good to be aware of the possibility. Here is the workaround:

```
// Returns false
Object.prototype.hasOwnProperty.call(voldemort, "ridikulus");
```

Finally, depending on the JavaScript engine, you may have trouble detecting the special `__proto__` property with either of these methods.

That's a brief introduction to the use of `in` and `hasOwnProperty`.

As always, thanks for reading!

Joshua Clanton
