DOM Attributes now on the prototype chain

By Paul Kinlan

Paul is a Developer Advocate

The Chrome team recently announced that we are moving DOM properties to the prototype chain. This change, implemented in Chrome 43 - (Beta as of mid April 2015) - brings Chrome more in line with the Web IDL Spec and other browsers' implementations, such as IE and Firefox. *Edit: clarified* Older WebKit based browsers, are currently not compatible with the spec, however Safari now is.

Note: The use of the word Attribute and Property are used interchangeably in this post, the ECMA Script spec defines 'Properties' that have 'Attributes'. A JS property *is* a 'WebIDL Attribute'. An Attribute in this article is not an HTML Attribute such as `class` on an image element.

The new behavior is positive in many ways. It:

- Improves compatibility across the web (IE and Firefox do this already) via compliance with the spec.
- Allows you to consistently and efficiently create getters/setters on every DOM Object.
- Increases the hackability of DOM programming. For example, it will enable you to implement polyfills that allow you to efficiently emulate functionality missing in some browsers and JavaScript libraries that override default DOM attribute behaviors.

For example, a hypothetical W3C specification includes some new functionality called **`isSuperContentEditable`** and the Chrome Browser doesn't implement it, but it is possible to "polyfill" or emulate the feature with a library. As the library developer, you would naturally want to use the **`prototype`** as follows to create an efficient polyfill:

```
Object.defineProperty(HTMLDivElement.prototype, "isSuperContentEditable", {
  get: function() { return true; },
  set: function() { /* some logic to set it up */ },
});
```

Prior to this change — for consistency with other DOM properties in Chrome — you would have had to create the new property on every instance, which for every **`HTMLDivElement`** on the page would be very inefficient.

These changes are important for consistency, performance and standardization of the web platform, yet they can cause some issues for developers. If you were relying on this behavior because of legacy compatibility between Chrome and WebKit we encourage you to check your site and see the summary of changes below.

Summary of changes

Using `hasOwnProperty` on a DOM Object instance will now return `false`

Sometimes developers will use `hasOwnProperty` to check for presence of an property on an object. This will no longer work as per the spec because DOM attributes are now part of the prototype chain and `hasOwnProperty` only inspects the current objects to see if it is defined on it.

Prior to and including Chrome 42 the following would return `true`.

```
> div = document.createElement("div");
> div.hasOwnProperty("isContentEditable");

true
```

In Chrome 43 onwards it will return `false`.

```
> div = document.createElement("div");
> div.hasOwnProperty("isContentEditable");

false
```

This now means if you want to check that `isContentEditable` is available on the element you will to have check the prototype on the HTMLElement object. For example `HTMLDivElement` inherits from `HTMLElement` which defines the `isContentEditable` property.

```
> HTMLElement.prototype.hasOwnProperty("isContentEditable");

true
```

You are not locked in to using `hasOwnProperty`. We recommend to use the much simpler `in` operand as this will check property on the entire prototype chain.

```
if("isContentEditable" in div) {
  // We have support!!
}
```

Object.getOwnPropertyDescriptor on DOM Object Instance will no longer return a property descriptor for Attributes.

If your site needs to get the property descriptor for an attribute on a DOM Object, you will now need to follow the prototype chain.

If you wanted to get the property description in Chrome 42 and earlier you would have done:

```
> Object.getOwnPropertyDescriptor(div, "isContentEditable");

Object {value: "", writable: true, enumerable: true, configurable: true}
```

Chrome 43 onwards will return `undefined` in this scenario.

```
> Object.getOwnPropertyDescriptor(div, "isContentEditable");

undefined
```

Which means to now get the property descriptor for the "isContentEditable" property you will need to follow the prototype chain as follows:

```
> Object.getOwnPropertyDescriptor(HTMLElement.prototype, "isContentEditable

Object {get: function, set: function, enumerable: false, configurable: false}
```

JSON.stringify will no longer serialize DOM Attributes

JSON.stringify doesn't serialize DOM properties that are on the prototype. For example, this can affect your site if you are trying to serialize an object such as Push Notification's PushSubscription.

Chrome 42 and earlier the following would have worked:

```
> JSON.stringify(subscription);

{
  "endpoint": "https://something",
  "subscriptionId": "SomeID"
}
```

Chrome 43 onwards will not serialize the properties that are on defined on the protoype and you will be returned an empty object.

```
> JSON.stringify(subscription);

{}
```

You will have to provide your own serialization method, for example you could do the following:

```
function stringifyDOMObject(object)
{
    function deepCopy(src) {
        if (typeof src != "object")
            return src;
        var dst = Array.isArray(src) ? [] : {};
        for (var property in src) {
            dst[property] = deepCopy(src[property]);
        }
        return dst;
    }
    return JSON.stringify(deepCopy(object));
}
var s = stringifyDOMObject(domObject);
```

Writing to read-only properties in strict mode will throw an error

Writing to read-only properties is supposed to throw an exception when you are using strict mode. For example, take the following:

```
function foo() {
  "use strict";
  var d = document.createElement("div");
  console.log(d.isContentEditable);
  d.isContentEditable = 1;
  console.log(d.isContentEditable);
}
```

Chrome 42 and earlier the function would have continued and silently carried on executing the function, although `isContentEditable` would have not been changed.

```
// Chrome 42 and earlier behavior
> foo();
```

```
false // isContentEditable
false // isContentEditable (after writing to read-only property)
```

Now in Chrome 43 and onwards there will be an exception thrown.

```
// Chrome 43 and onwards behavior
> foo();

false
Uncaught TypeError: Cannot set property isContentEditable of #<HTMLElement> which
```

I have a problem, what should I do?

Follow the guidance, or leave a comment below and let's talk.

I have seen a site with a problem, what should I do?

Great question. Most issues with sites will be based on the fact a site has chosen to do Attribute presence detection with the `getOwnProperty` method, this is mostly done when a site owner has only targeted older WebKit browsers. There are a couple of things that a developer can do:

- File an issue about the affected site on our (Chrome's) issue tracker
- File an issue on WebKit radar and reference https://bugs.webkit.org/show_bug.cgi?id=49739

I am generally interested in following this change

- Original bug from 2010: https://code.google.com/p/chromium/issues/detail?id=43394 - note: has most of the work on it.
- Code Review for commit

Last updated July 2, 2018.