7.11 Browser differences: JavaScript

Understanding browser differences

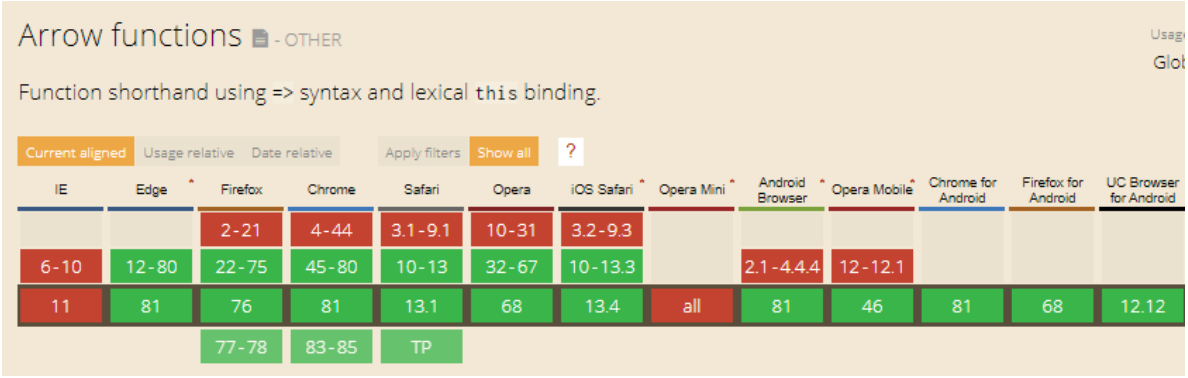
While W3C, WHATWG, and Ecma International define standards for HTML, CSS, and JavaScript, browsers may implement the standards differently for multiple reasons.

- A browser may have been released before a feature was standardized by the W3C or WHATWG, or the standard may have changed a particular specification. So, the browser vendor may not yet have added the new functionality.
- A browser vendor was helping WHATWG develop the standard API and may have started with a different proposed implementation than the final implementation. Typically, future versions of the browser will implement the final specification, but older versions may support the initial implementation.
- A browser vendor created a new API without input from WHATWG to differentiate the browser from other vendors' browsers. Browser differentiation was a prominent feature of the first browser wars, but non-standard browsers are now much less accepted by web developers and users. Browsers are expected to conform to standards.
- For open source browsers, the browser development team typically consists of volunteers who may not yet have been able to make the browser conform to the latest standards.
- A browser may have a bug causing the browser to interpret the code differently than the standard details.

In the early 2000's, web developers were expected to know the major differences between browsers, so that web applications would look the same in all browsers. Many articles were written to help developers, such as 7 JavaScript Differences Between Firefox and IE by Louis Lazaris. As developers began to expect standards conformance from browser vendors, online resources were created to help developers utilize standards.

CanIUse is a website that tracks which features are actively used on the web. The website also tracks which features are supported by all versions of the major browsers and displays those features in a matrix to show when those features were first supported. CanIUse uses statistics from various sources to indicate the percentage of web users accessing the web with each browser version. CanIUse also determines the percentage of web users with browsers capable of using a particular feature. After the percentage reaches some minimum threshold, a developer can determine that the new feature is worth using because enough people will benefit and not too many will be "harmed."

Figure 7.11.1: CanIUse: Arrow functions.



Source: CanIUse

Feedback?

PARTICIPATION ACTIVITY 7.11.1: JavaScript support in browsers.

Guess the answer to the following questions to get a feel for JavaScript feature support in browsers. The degree of support is sometimes higher and sometimes lower than might be expected.

1) What percentage of the ECMAScript 5 standard (2009) does Internet Explorer 10 (2012) support?

- ○ 48%
- ◉ 99%

Correct

Internet Explorer 10 supported all but one feature of ECMAScript 5.

2) What percentage of the ECMAScript 6 standard (2015) does Internet Explorer 10 (2012) support?

- ◉ 5%
- ○ 60%

Correct

ECMAScript 6 was finalized in 2015, three years *after* Internet Explorer 10 was released.

3) Which mobile operating system browser supported the highest percentage of ECMAScript 6 features in 2016, one year after ECMAScript 6 was standardized?

- Android 5.1
- iOS 9

Correct

iOS 9 supported 53% of ECMAScript 6 features.

Feedback?

Polyfills

A ***polyfill*** is JavaScript code that provides missing standard functionality for older browsers. A polyfill typically checks for the presence of a feature in the browser and uses the built in version if available. Otherwise, the polyfill executes JavaScript code that implements similar functionality. Common polyfill uses include form widgets, video and audio, geo-location, and WebGL.

Note

A polyfill may not behave the same as a native implementation by a browser, but the benefit of being able to use otherwise unsupported features generally outweighs slight differences.

Example 7.11.1: An HTML date polyfill.

If the date input element is not supported by a browser, the browser treats the date input box as an ordinary text input field.

```
<form>
  Select an appointment date:<br>
  <input type="date" name="appointmentDate"><br>
  <input type="submit">
</form>
```

Date input rendered by browser supporting (left) and not supporting (right) date input

A polyfill for supporting the date input can use Modernizr and the jQuery datepicker by:

1. Loading the jQuery and Modernizr libraries.
2. Calling a jQuery function that:
    - Uses Modernizr to determine whether the date input is supported
    - Uses the jQuery datepicker if needed
3. The original input form is unmodified by the polyfill.

```
<link href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.0/jquer
ui.min.css" rel="stylesheet">
<script
src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.mi
</script>
<script src="https://code.jquery.com/jquery-3.1.0.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.0/jquery-ui.min.js"></scrip

<script>
$(function() {
   if (!Modernizr.inputtypes['date']) {
      $('input[type=date]').datepicker({
         dateFormat: 'mm-dd-yy'
      });
   }
});
</script>

<form>
   Select an appointment date:<br>
   <input type="date" name="appointmentDate"><br>
   <input type="submit">
</form>
```

Polyfill for date input using Modernizr and the jQuery date picker

Feedback?

Some browsers may implement JavaScript DOM objects differently, such as by not supporting a standard method for the object. Ex: Internet Explorer 8 does not support the Date.now() method. A JavaScript polyfill can be used to add unsupported methods to existing objects.

Example 7.11.2: Creating a polyfill for the Date.now() method.

To support the Date.now() method in an older browser, a web developer can add a now() method to the global Date object.

```
if (!Date.now) {
   Date.now = function() {
      return new Date().getTime();
   }
}
```

The code checks whether `Date.now` method is defined. If the now method is not defined, a now attribute is created in the Date object and set to reference a programmer defined function.

Alternatively, the Date.now() polyfill can be written more succinctly as:

```
Date.now = (Date.now || function() {
      return new Date().getTime();
});
```

`Date.now` is set to the result of evaluating `(Date.now || function() {...}`. If `Date.now` is defined and consequently truthy, then `Date.now` is set to the current value of `Date.now`. Otherwise, if the `Date.now` method is not defined and consequently falsy, then a now attribute is created in Date and set to reference the function defined to the right of the || operator.

Feedback?

PARTICIPATION ACTIVITY 7.11.2: Review browser JavaScript differences.

1) A missing JavaScript DOM object property is relatively easy to replace.
   - (•) True
   - ( ) False

Correct

Because JavaScript allows object properties, including methods, to be added after an object has been created, missing properties can be easily added later.

2) A polyfill is JavaScript code that provides missing standard functionality for older browsers.

- (●) True
- ( ) False

Correct

Polyfills are JavaScript code used to solve compatibility issues for HTML, CSS, and JavaScript.

3) Modernizr is a good polyfill for adding missing functionality to older browsers.

- ( ) True
- (●) False

Correct

Modernizr is a JavaScript library used to detect which features are supported but does not add missing functionality to older browsers.

Feedback?

Exploring further:

- Modernizr on GitHub
- Polyfills from MDN
- caniuse.com
- JavaScript compatibility table

How was this section? **Provide section feedback**