Array.prototype.filter()

The **`filter()`** method creates a new array with all elements that pass the test implemented by the provided function.

Syntax

```
arr.filter(callback[, thisArg])
```

Parameters

`callback`

Function to test each element of the array. Invoked with arguments `(element, index, array)`. Return `true` to keep the element, `false` otherwise.

`thisArg`

Optional. Value to use as `this` when executing `callback`.

Description

`filter()` calls a provided `callback` function once for each element in an array, and constructs a new array of all the values for which `callback` returns a `true` value or a value that coerces to `true`. `callback` is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values. Array elements which do not pass the `callback` test are simply skipped, and are not included in the new array.

`callback` is invoked with three arguments:

1. the value of the element
2. the index of the element
3. the Array object being traversed

If a `thisArg` parameter is provided to `filter`, it will be passed to `callback` when invoked, for use as its `this` value. Otherwise, the value `undefined` will be passed for use as its `this` value. The `this` value ultimately observable by `callback` is determined according to the usual rules for determining the `this` seen by a function.

`filter()` does not mutate the array on which it is called.

The range of elements processed by `filter()` is set before the first invocation of `callback`. Elements which are appended to the array after the call to `filter()` begins will not be visited by `callback`. If existing elements of the array are changed, or deleted, their value as passed to `callback` will be the value at the time `filter()` visits them; elements that are deleted are not visited.

Examples

Example: Filtering out all small values

The following example uses `filter()` to create a filtered array that has all elements with values less than 10 removed.

```
function isBigEnough(value) {
  return value >= 10;
}
var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
// filtered is [12, 130, 44]
```

Example: Filtering invalid entries from JSON

The following example uses `filter()` to create a filtered json of all elements with non-zero, numeric `id`.

```
var arr = [
  { id: 15 },
  { id: -1 },
  { id: 0 },
  { id: 3 },
  { id: 12.2 },
  { },
  { id: null },
  { id: NaN },
  { id: 'undefined' }
];

var invalidEntries = 0;

function filterByID(obj) {
  if ('id' in obj && typeof(obj.id) === 'number' && !isNaN(obj.id)) {
    return true;
  } else {
    invalidEntries++;
    return false;
  }
}

var arrByID = arr.filter(filterByID);

console.log('Filtered Array\n', arrByID);
// [{ id: 15 }, { id: -1 }, { id: 0 }, { id: 3 }, { id: 12.2 }]

console.log('Number of Invalid Entries = ', invalidEntries);
// 4
```

Polyfill

`filter()` was added to the ECMA-262 standard in the 5th edition; as such it may not be present in all implementations of the standard. You can work around this by inserting the following code at the beginning of your scripts, allowing use of `filter()` in ECMA-262 implementations which do not natively support it. This algorithm is exactly the one specified in ECMA-262, 5th edition, assuming that `fn.call` evaluates to the original value of `Function.prototype.call()`, and that `Array.prototype.push()` has its original value.

```
if (!Array.prototype.filter) {
  Array.prototype.filter = function(fun/*, thisArg*/) {
    'use strict';

    if (this === void 0 || this === null) {
      throw new TypeError();
    }

    var t = Object(this);
    var len = t.length >>> 0;
    if (typeof fun !== 'function') {
      throw new TypeError();
    }

    var res = [];
    var thisArg = arguments.length >= 2 ? arguments[1] : void 0;
    for (var i = 0; i < len; i++) {
      if (i in t) {
        var val = t[i];

        // NOTE: Technically this should Object.defineProperty at
        //       the next index, as push can be affected by
        //       properties on Object.prototype and Array.prototype.
        //       But that method's new, and collisions should be
        //       rare, so use the more-compatible alternative.
        if (fun.call(thisArg, val, i, t)) {
          res.push(val);
        }
      }
```

```
    }

    return res;
  };
}
```

Specifications

Specification	Status	Comment
ECMAScript 5.1 (ECMA-262) The definition of 'Array.prototype.filter' in that specification.	Standard	Initial definition. Implemented in JavaScript 1.6.
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.filter' in that specification.	Standard	

Browser compatibility

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	1.5 (1.8)	9	(Yes)	(Yes)

See also

- `Array.prototype.forEach()`
- `Array.prototype.every()`
- `Array.prototype.some()`
- `Array.prototype.reduce()`