当你不需要 Jquery 时，你是否还记得原生的 API?

前端发展很快，现代浏览器原生 API 已经足够好用。现代前端项目的开发中，我们并不需要为了操作 DOM。由于 React、Angular、Vue 等框架的流行，直接操作 DOM 不再是好的模式，jQuery 使用场景大大减少。（但是不是 Jquery 就没有用了呢？，并不是，本篇文章就不过多的阐述与说明）本文章总结了大部分 jQuery API 替代的方法，暂时只支持 IE10+ 以上浏览器。

Query Selector

常用的 class、id、属性 选择器都可以使用 document.querySelector 或 document.querySelectorAll 替代。
区别是

document.querySelector 返回第一个匹配的 Element
document.querySelectorAll 返回所有匹配的 Element 组成的 NodeList。它可以通过 [].slice.call() 把它转成 Array.
如果匹配不到任何 Element，jQuery 返回空数组 []，
但 document.querySelector 返回 null，注意空指针异常。当找不到时，也可以使用 || 设置默认的值，如 document.querySelectorAll(selector) || []
注意：document.querySelector 和 document.querySelectorAll 性能很差。如果想提高性能，尽量使用 document.getElementById、
document.getElementsByClassName 或 document.getElementsByTagName。

1.0 选择器查询

```
// jQuery
$('selector');

// Native
document.querySelectorAll('selector');
```

1.1 class 查询

```
// jQuery
```

```
$('.class');

// Native
document.querySelectorAll('.class');

// or
document.getElementsByClassName('class');
```

1.2 id 查询

```
// jQuery
$('#id');

// Native
document.querySelector('#id');

// or
document.getElementById('id');
```

1.3 属性查询

```
// jQuery
$('a[target=_blank]');

// Native
document.querySelectorAll('a[target=_blank]');
```

1.4 后代查询

```
// jQuery
$el.find('li');

// Native
el.querySelectorAll('li');
```

1.5 兄弟及上下元素

兄弟元素

```
// jQuery
$el.siblings();

// Native - latest, Edge13+
[...el.parentNode.children].filter((child) =>
  child !== el
);
// Native (alternative) - latest, Edge13+
Array.from(el.parentNode.children).filter((child) =>
```

```
  child !== el
);
// Native - IE10+
Array.prototype.filter.call(el.parentNode.children, (child) =>
  child !== el
);
```

上一个元素

```
// jQuery
$el.prev();

// Native
el.previousElementSibling;
```

下一个元素

```
// next
$el.next();

// Native
el.nextElementSibling;
```

1.6 Closest

Closest 获得匹配选择器的第一个祖先元素，从当前元素开始沿 DOM 树向上。

```
// jQuery
$el.closest(queryString);

// Native - Only latest, NO IE
el.closest(selector);

// Native - IE10+
function closest(el, selector) {
  const matchesSelector = el.matches || el.webkitMatchesSelector ||
el.mozMatchesSelector || el.msMatchesSelector;

  while (el) {
    if (matchesSelector.call(el, selector)) {
      return el;
    } else {
      el = el.parentElement;
    }
  }
  return null;
}
```

1.7 Parents Until

获取当前每一个匹配元素集的祖先，不包括匹配元素的本身。

```
// jQuery
$el.parentsUntil(selector, filter);

// Native
function parentsUntil(el, selector, filter) {
  const result = [];
  const matchesSelector = el.matches || el.webkitMatchesSelector ||
el.mozMatchesSelector || el.msMatchesSelector;

  // match start from parent
  el = el.parentElement;
  while (el && !matchesSelector.call(el, selector)) {
    if (!filter) {
      result.push(el);
    } else {
      if (matchesSelector.call(el, filter)) {
        result.push(el);
      }
    }
    el = el.parentElement;
  }
  return result;
}
```

1.8 Form

Input/Textarea

```
// jQuery
$('#my-input').val();

// Native
document.querySelector('#my-input').value;
```

获取 e.currentTarget 在 .radio 中的数组索引

```
// jQuery
$('.radio').index(e.currentTarget);

// Native
Array.prototype.indexOf.call(document.querySelectorAll('.radio'),
e.currentTarget);
```

1.9 Iframe Contents

jQuery 对象的 iframe contents() 返回的是 iframe 内的 document

Iframe contents

```
// jQuery
$iframe.contents();

// Native
iframe.contentDocument;
```

Iframe Query

```
// jQuery
$iframe.contents().find('.css');

// Native
iframe.contentDocument.querySelectorAll('.css');
```

1.10 获取 body

```
// jQuery
$('body');

// Native
document.body;
```

1.11 获取或设置属性

获取属性

```
// jQuery
$el.attr('foo');

// Native
el.getAttribute('foo');
```

设置属性

```
// jQuery, note that this works in memory without change the DOM
$el.attr('foo', 'bar');

// Native
el.setAttribute('foo', 'bar');
```

获取 data- 属性

```
// jQuery
```

```
$el.data('foo');

// Native (use `getAttribute`)
el.getAttribute('data-foo');

// Native (use `dataset` if only need to support IE 11+)
el.dataset['foo'];
```

CSS & Style

2.1 CSS

Get style

```
// jQuery
$el.css("color");

// Native
// 注意：此处为了解决当 style 值为 auto 时，返回 auto 的问题
const win = el.ownerDocument.defaultView;

// null 的意思是不返回伪类元素
win.getComputedStyle(el, null).color;
```

Set style

```
// jQuery
$el.css({ color: "#ff0011" });

// Native
el.style.color = '#ff0011';
```

Get/Set Styles

注意，如果想一次设置多个 style，可以参考 oui-dom-utils 中 setStyles 方法

Add class

```
// jQuery
$el.addClass(className);

// Native
el.classList.add(className);
```

Remove class

```
// jQuery
$el.removeClass(className);

// Native
el.classList.remove(className);
```

has class

```
// jQuery
$el.hasClass(className);

// Native
el.classList.contains(className);
```

Toggle class

```
// jQuery
$el.toggleClass(className);

// Native
el.classList.toggle(className);
```

2.2 Width & Height

Width 与 Height 获取方法相同，下面以 Height 为例：

Window height

```
// window height
$(window).height();

// 含 scrollbar
window.document.documentElement.clientHeight;

// 不含 scrollbar，与 jQuery 行为一致
window.innerHeight;
```

Document height

```
// jQuery
$(document).height();

// Native
const body = document.body;
const html = document.documentElement;
const height = Math.max(
```

```
  body.offsetHeight,
  body.scrollHeight,
  html.clientHeight,
  html.offsetHeight,
  html.scrollHeight
);
Element height

// jQuery
$el.height();

// Native
function getHeight(el) {
  const styles = this.getComputedStyle(el);
  const height = el.offsetHeight;
  const borderTopWidth = parseFloat(styles.borderTopWidth);
  const borderBottomWidth = parseFloat(styles.borderBottomWidth);
  const paddingTop = parseFloat(styles.paddingTop);
  const paddingBottom = parseFloat(styles.paddingBottom);
  return height - borderBottomWidth - borderTopWidth - paddingTop -
paddingBottom;
}

// 精确到整数 (border-box 时为 height - border 值, content-box 时为
height + padding 值)
el.clientHeight;

// 精确到小数 (border-box 时为 height 值, content-box 时为 height +
padding + border 值)
el.getBoundingClientRect().height;
```

2.3 Position & Offset

Position

获得匹配元素相对父元素的偏移

```
// jQuery
$el.position();

// Native
{ left: el.offsetLeft, top: el.offsetTop }
```

Offset

获得匹配元素相对文档的偏移

```
// jQuery
$el.offset();

// Native
function getOffset (el) {
  const box = el.getBoundingClientRect();

  return {
    top: box.top + window.pageYOffset -
document.documentElement.clientTop,
    left: box.left + window.pageXOffset -
document.documentElement.clientLeft
  }
}
```

2.4 Scroll Top

获取元素滚动条垂直位置。

```
// jQuery
$(window).scrollTop();

// Native
(document.documentElement && document.documentElement.scrollTop) ||
document.body.scrollTop;
```


DOM Manipulation

3.1 Remove

从 DOM 中移除元素。

```
// jQuery
$el.remove();

// Native
el.parentNode.removeChild(el);
```

3.2 Text

Get text

返回指定元素及其后代的文本内容。

```
// jQuery
$el.text();

// Native
el.textContent;
```

Set text

设置元素的文本内容。

```
// jQuery
$el.text(string);

// Native
el.textContent = string;
```

3.3 HTML

Get HTML

```
// jQuery
$el.html();

// Native
el.innerHTML;
```

Set HTML

```
// jQuery
$el.html(htmlString);

// Native
el.innerHTML = htmlString;
```

3.4 Append

Append 插入到子节点的末尾

```
// jQuery
$el.append("<div id='container'>hello</div>");

// Native (HTML string)
el.insertAdjacentHTML('beforeend', '<div id="container">Hello
World</div>');

// Native (Element)
el.appendChild(newEl);
```

3.5 Prepend

```
// jQuery
$el.prepend("<div id='container'>hello</div>");

// Native (HTML string)
el.insertAdjacentHTML('afterbegin', '<div id="container">Hello
World</div>');

// Native (Element)
el.insertBefore(newEl, el.firstChild);
```

3.6 insertBefore

在选中元素前插入新节点

```
// jQuery
$newEl.insertBefore(queryString);

// Native (HTML string)
el.insertAdjacentHTML('beforebegin ', '<div id="container">Hello
World</div>');

// Native (Element)
const el = document.querySelector(selector);
if (el.parentNode) {
  el.parentNode.insertBefore(newEl, el);
}
```

3.7 insertAfter

在选中元素后插入新节点

```
// jQuery
$newEl.insertAfter(queryString);

// Native (HTML string)
el.insertAdjacentHTML('afterend', '<div id="container">Hello
World</div>');

// Native (Element)
const el = document.querySelector(selector);
if (el.parentNode) {
  el.parentNode.insertBefore(newEl, el.nextSibling);
}
```

3.8 is

如果匹配查询选择器则 true

```
// jQuery - Notice `is` also work with `function` or `elements` which is
not concerned here
$el.is(selector);

// Native
el.matches(selector);
```

3.9 clone

创建元素的深度拷贝

```
// jQuery
$el.clone();

// Native
el.cloneNode();

// For Deep clone , set param as `true`
```

3.10 empty

移除所有子节点

```
// jQuery
$el.empty();

// Native
el.innerHTML = '';
```

3.11 wrap

使用 HTML 结构包装每个元素

```
// jQuery
$('.inner').wrap('<div class="wrapper"></div>');

// Native
Array.prototype.forEach.call(document.querySelectorAll('.inner'), (el)
=> {
  const wrapper = document.createElement('div');
  wrapper.className = 'wrapper';
  el.parentNode.insertBefore(wrapper, el);
  el.parentNode.removeChild(el);
  wrapper.appendChild(el);
```

```
});
```

3.12 unwrap

从 DOM 中移除匹配元素的父集

```
// jQuery
$('.inner').unwrap();

// Native
Array.prototype.forEach.call(document.querySelectorAll('.inner'), (el)
=> {
  Array.prototype.forEach.call(el.childNodes, (child) => {
    el.parentNode.insertBefore(child, el);
  });
  el.parentNode.removeChild(el);
});
```

3.13 replaceWith

使用提供的新内容替换匹配元素的内容

```
// jQuery
$('.inner').replaceWith('<div class="outer"></div>');

// Native
Array.prototype.forEach.call(document.querySelectorAll('.inner'), (el)
=> {
  const outer = document.createElement('div');
  outer.className = 'outer';
  el.parentNode.insertBefore(outer, el);
  el.parentNode.removeChild(el);
});
```

用指定的元素替换被选的元素

```
//jQuery
$('.inner').replaceWith('<div class="outer"></div>');

//Native
Array.prototype.forEach.call(document.querySelectorAll('.inner'),(el)
=> {
  const outer = document.createElement("div");
  outer.className = "outer";
  el.parentNode.insertBefore(outer, el);
  el.parentNode.removeChild(el);
});
```

Ajax

Fetch API 是用于替换 XMLHttpRequest 处理 ajax 的新标准，Chrome 和 Firefox 均支持，旧浏览器可以使用 polyfills 提供支持。

IE9+ 请使用 github/fetch，IE8+ 请使用 fetch-ie8，JSONP 请使用 fetch-jsonp。

4.1 从服务器读取数据并替换匹配元素的内容。

```
// jQuery
$(selector).load(url, completeCallback)

// Native
fetch(url).then(data => data.text()).then(data => {
  document.querySelector(selector).innerHTML = data
}).then(completeCallback)
```

Events
完整地替代命名空间和事件代理，链接到 github.com/oneuijs/oui…

5.0 Document ready by DOMContentLoaded

```
// jQuery
$(document).ready(eventHandler);

// Native
```

```
// 检测 DOMContentLoaded 是否已完成
if (document.readyState === 'complete' || document.readyState !==
'loading') {
  eventHandler();
} else {
  document.addEventListener('DOMContentLoaded', eventHandler);
}
```

5.1 使用 on 绑定事件件

```
// jQuery
$el.on(eventName, eventHandler);

// Native
el.addEventListener(eventName, eventHandler);
```

5.2 使用 off 解绑事件件

```
// jQuery
$el.off(eventName, eventHandler);

// Native
el.removeEventListener(eventName, eventHandler);
```

5.3 Trigger

```
// jQuery
$(el).trigger('custom-event', {key1: 'data'});

// Native
if (window.CustomEvent) {
  const event = new CustomEvent('custom-event', {detail: {key1:
'data'}});
} else {
  const event = document.createEvent('CustomEvent');
  event.initCustomEvent('custom-event', true, true, {key1: 'data'});
}

el.dispatchEvent(event);
```

Utilities

大部分实用工具都能在 native API 中找到. 其他高级功能可以选用专注于该领域的稳定性和性能都更好的库来代替, 推荐 lodash。

6.1 基本工具

isArray
检测参数是不是数组。

```
// jQuery
$.isArray(range);

// Native
Array.isArray(range);
```

isWindow
检测参数是不是 window。

```
// jQuery
$.isWindow(obj);

// Native
function isWindow(obj) {
  return obj !== null && obj !== undefined && obj === obj.window;
}
```

inArray
在数组中搜索指定值并返回索引 (找不到则返回 -1)。

```
// jQuery
$.inArray(item, array);

// Native
array.indexOf(item) > -1;

// ES6-way
array.includes(item);
```

isNumeric
检测传入的参数是不是数字。 Use typeof to decide the type or the type example for better accuracy.

```
// jQuery
$.isNumeric(item);

// Native
function isNumeric(value) {
  var type = typeof value;

  return (type === 'number' || type === 'string') && !Number.isNaN(value - Number.parseFloat(value));
```

```
}
```

isFunction
检测传入的参数是不是 JavaScript 函数对象。

```
// jQuery
$.isFunction(item);

// Native
function isFunction(item) {
  if (typeof item === 'function') {
    return true;
  }
  var type = Object.prototype.toString(item);
  return type === '[object Function]' || type === '[object
GeneratorFunction]';
}
```

isEmptyObject
检测对象是否为空（包括不可枚举属性).

```
// jQuery
$.isEmptyObject(obj);

// Native
function isEmptyObject(obj) {
  return Object.keys(obj).length === 0;
}
```

isPlainObject
检测是不是扁平对象（使用 “{}” 或 “new Object” 创建).

```
// jQuery
$.isPlainObject(obj);

// Native
function isPlainObject(obj) {
  if (typeof (obj) !== 'object' || obj.nodeType || obj !== null && obj !==
undefined && obj === obj.window) {
    return false;
  }

  if (obj.constructor &&
      !Object.prototype.hasOwnProperty.call(obj.constructor.prototype,
'isPrototypeOf')) {
    return false;
  }
```

```
  return true;
}
```

extend

合并多个对象的内容到第一个对象。 object.assign 是 ES6 API，也可以使用 polyfill。

```
// jQuery
$.extend({}, defaultOpts, opts);

// Native
Object.assign({}, defaultOpts, opts);
```

trim

移除字符串头尾空白。

```
// jQuery
$.trim(string);

// Native
string.trim();
```

map

将数组或对象转化为包含新内容的数组。

```
// jQuery
$.map(array, (value, index) => {
});

// Native
array.map((value, index) => {
});
```

each

轮询函数，可用于平滑的轮询对象和数组。

```
// jQuery
$.each(array, (index, value) => {
});

// Native
array.forEach((value, index) => {
});
```

grep

找到数组中符合过滤函数的元素。

```
// jQuery
```

```
$.grep(array, (value, index) => {
});

// Native
array.filter((value, index) => {
});
```

type

检测对象的 JavaScript [Class] 内部类型。

```
// jQuery
$.type(obj);

// Native
function type(item) {
  const reTypeOf = /(?:^\[object\s(.*?)\]$)/;
  return Object.prototype.toString.call(item)
    .replace(reTypeOf, '$1')
    .toLowerCase();
}
```

merge

合并第二个数组内容到第一个数组。

```
// jQuery
$.merge(array1, array2);

// Native
// But concat function doesn't remove duplicate items.
function merge(...args) {
  return [].concat(...args)
}
```

now

返回当前时间的数字呈现。

```
// jQuery
$.now();

// Native
Date.now();
```

proxy

传入函数并返回一个新函数，该函数绑定指定上下文。

```
// jQuery
$.proxy(fn, context);
```

```
// Native
fn.bind(context);
```

makeArray
类数组对象转化为真正的 JavaScript 数组。

```
// jQuery
$.makeArray(arrayLike);

// Native
Array.prototype.slice.call(arrayLike);

// ES6-way
Array.from(arrayLike);
```

6.2 包含

检测 DOM 元素是不是其他 DOM 元素的后代.

```
// jQuery
$.contains(el, child);

// Native
el !== child && el.contains(child);
```

6.3 Globaleval

全局执行 JavaScript 代码。

```
// jQuery
$.globaleval(code);

// Native
function Globaleval(code) {
  const script = document.createElement('script');
  script.text = code;

  document.head.appendChild(script).parentNode.removeChild(script);
}

// Use eval, but context of eval is current, context of $.Globaleval is
global.
eval(code);
```

6.4 解析

parseHTML
解析字符串为 DOM 节点数组.

```
// jQuery
$.parseHTML(htmlString);

// Native
function parseHTML(string) {
  const context = document.implementation.createHTMLDocument();

  // Set the base href for the created document so any parsed elements
with URLs
  // are based on the document's URL
  const base = context.createElement('base');
  base.href = document.location.href;
  context.head.appendChild(base);

  context.body.innerHTML = string;
  return context.body.children;
}
```

parseJSON

传入格式正确的 JSON 字符串并返回 JavaScript 值.

```
// jQuery
$.parseJSON(str);

// Native
JSON.parse(str);
```


Promises

Promise 代表异步操作的最终结果。jQuery 用它自己的方式处理 promises，原生 JavaScript 遵循 Promises/A+ 标准实现了最小 API 来处理 promises。

7.1 done, fail, always

done 会在 promise 解决时调用，fail 会在 promise 拒绝时调用，always 总会调用。

```
// jQuery
$promise.done(doneCallback).fail(failCallback).always(alwaysCallback)

// Native
```

```
promise.then(doneCallback, failCallback).then(alwaysCallback,
alwaysCallback)
```

7.2 when

when 用于处理多个 promises。当全部 promises 被解决时返回，当任一 promise 被拒绝时拒绝。

```
// jQuery
$.when($promise1, $promise2).done((promise1Result, promise2Result) => {
});

// Native
Promise.all([$promise1, $promise2]).then([promise1Result,
promise2Result] => {});
```

7.3 Deferred

Deferred 是创建 promises 的一种方式。

```
// jQuery
function asyncFunc() {
  const defer = new $.Deferred();
  setTimeout(() => {
    if(true) {
      defer.resolve('some_value_computed_asynchronously');
    } else {
      defer.reject('failed');
    }
  }, 1000);

  return defer.promise();
}

// Native
function asyncFunc() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (true) {
        resolve('some_value_computed_asynchronously');
      } else {
        reject('failed');
      }
    }, 1000);
  });
}
```

```
// Deferred way
function defer() {
  const deferred = {};
  const promise = new Promise((resolve, reject) => {
    deferred.resolve = resolve;
    deferred.reject = reject;
  });

  deferred.promise = () => {
    return promise;
  };

  return deferred;
}

function asyncFunc() {
  const defer = defer();
  setTimeout(() => {
    if(true) {
      defer.resolve('some_value_computed_asynchronously');
    } else {
      defer.reject('failed');
    }
  }, 1000);

  return defer.promise();
}
```


Animation

8.1 Show & Hide

```
// jQuery
$el.show();
$el.hide();

// Native
// 更多 show 方法的细节详见
https://github.com/oneuijs/oui-dom-utils/blob/master/src/index.js#L363
el.style.display = ''|'inline'|'inline-block'|'inline-table'|'block';
```

```
el.style.display = 'none';
```

8.2 Toggle

显示或隐藏元素。

```
// jQuery
$el.toggle();

// Native
if (el.ownerDocument.defaultView.getComputedStyle(el, null).display
=== 'none') {
  el.style.display =
''|'inline'|'inline-block'|'inline-table'|'block';
} else {
  el.style.display = 'none';
}
```

8.3 FadeIn & FadeOut

```
// jQuery
$el.fadeIn(3000);
$el.fadeOut(3000);

// Native
el.style.transition = 'opacity 3s';
// fadeIn
el.style.opacity = '1';
// fadeOut
el.style.opacity = '0';
```

8.4 FadeTo

调整元素透明度。

```
// jQuery
$el.fadeTo('slow',0.15);
// Native
el.style.transition = 'opacity 3s'; // 假设 'slow' 等于 3 秒
el.style.opacity = '0.15';
```

8.5 FadeToggle

动画调整透明度用来显示或隐藏。

```
// jQuery
$el.fadeToggle();
```

```
// Native
el.style.transition = 'opacity 3s';
const { opacity } = el.ownerDocument.defaultView.getComputedStyle(el,
null);
if (opacity === '1') {
  el.style.opacity = '0';
} else {
  el.style.opacity = '1';
}
```

8.6 SlideUp & SlideDown

```
// jQuery
$el.slideUp();
$el.slideDown();

// Native
const originHeight = '100px';
el.style.transition = 'height 3s';
// slideUp
el.style.height = '0px';
// slideDown
el.style.height = originHeight;
```

8.7 SlideToggle

滑动切换显示或隐藏。

```
// jQuery
$el.slideToggle();

// Native
const originHeight = '100px';
el.style.transition = 'height 3s';
const { height } = el.ownerDocument.defaultView.getComputedStyle(el,
null);
if (parseInt(height, 10) === 0) {
  el.style.height = originHeight;
}
else {
 el.style.height = '0px';
}
```

8.8 Animate

执行一系列 CSS 属性动画。

```
// jQuery
$el.animate({ params }, speed);

// Native
el.style.transition = 'all ' + speed;
Object.keys(params).forEach((key) =>
  el.style[key] = params[key];
)
```