[【前端性能】必须要掌握的原生JS实现JQuery](http://www.cnblogs.com/coco1s/p/4484238.html)

http://www.cnblogs.com/coco1s/p/4484238.html

很多时候，我们经常听见有人说jquery有多快多快。在这个各种类库满天飞的时候，不得不说的是，能有原生JS快吗？

是的，明显原生JS要更快，因为诸如JQuery这样的库必须要兼容各种浏览器和低版本和许多其他的东西，考虑通用性必然会导致性能的损耗。

当然，我不是提倡写代码纯用原生JS实现，我记得淘宝玉伯曾经说的一句话“框架能够让我们走的更快，但只有了解原生的JS才能让我们走的更远”。

一些感（fei）想（hua）：

* 原生的js，好比全真教的武功，一步步从基础开始（先练气再御剑），很长一段时间内和jquery有很大差距，掌握以后发现jquery只不过是另外一种武功，看一遍既会。且当学原生到一定程度之后，可以自创武功。但原生见效很慢，属于前期慢后期快，成长性高。
* jquery，好比华山派的剑宗，直接取实用的部分（剑），拿来即用，实用快速为主（剑），原理部分的原生为辅（气）。jquery见效较快，属于前期较快后期较慢，没有气的支持，成长会越来越慢
* 网上找代码，好比赵敏找六大门派高手偷绝学，直接对症下药，需要什么拿什么用什么，只注重伤敌（结果），无所谓剑与气。见效奇快，属于前期快，后期慢。如果事后不研究原理，完全不会有成长

废话不多说，下面的代码示例是一些流行的常用的JQuery功能的原生JS实现。

**jQuery 选择器的原生实现**

便捷的找到我们想要的DOM元素是JQuery的核心功能，JQuery选择器的强大无需赘言。通过传递一个查询字符串给CSS选择器，它将会在DOM元素中检索所有的匹配。

然而在大多数的情况下，我们可以使用相同的代码量用原生JS轻易实现。

**1.获取页面所有的 div**

|  |  |
| --- | --- |
| 1  2  3  4  5 | /\* jQuery \*/  $("div")    /\* native equivalent \*/  document.getElementsByTagName("div") |

**2.获取某类名相同的一群元素**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | /\* jQuery \*/  $(".my-class")    /\* native equivalent \*/  document.querySelectorAll(".my-class")    /\* FASTER native equivalent \*/  document.getElementsByClassName("my-class") |

**3.更复杂的一些选择器**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5 | /\* jQuery \*/  $(".my-class li:first-child")    /\* native equivalent \*/  document.querySelectorAll(".my-class li:first-child") | |
| 1  2  3  4  5 | | /\* jQuery \*/  $(".my-class").get(0)    /\* native equivalent \*/  document.querySelector(".my-class") |

**jQuery DOM 操作的原生实现**

JQuery另一大频繁被使用到的功能就是操作DOM元素，诸如插入或删除一个元素。当然，如果使用原生JS实现这些功能，代码量肯定是会有所增加的，不过我们也可以将这些功能封装成函数，下面是一些常用的DOM操作的原生JS实现。

**1.插入HTML元素**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | /\* jQuery \*/  $(document.body).append("<div id='myDiv'><img src='im.gif'/></div>");    /\* CRAPPY native equivalent \*/  document.body.innerHTML += "<div id='myDiv'><img src='im.gif'/></div>";    /\* MUCH BETTER native equivalent \*/  var frag = document.createDocumentFragment();    var myDiv = document.createElement("div");  myDiv.id = "myDiv";    var im = document.createElement("img");  im.src = "im.gif";    myDiv.appendChild(im);  frag.appendChild(myDiv);    document.body.appendChild(frag); |

**jQuery CSS 操作的原生实现**

在JQuery中可以轻松实现对css的操作，增加属性、删除属性或是检测是否存在某个类。那么你是否觉得使用原生JS实现会很麻烦呢？其实不然，因为有**classList**。下面是一些关于JQuery css操作的JS原生实现。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | // get reference to DOM element  var el = document.querySelector(".main-content");    //----Adding a class------    /\* jQuery \*/  $(el).addClass("someClass");    /\* native equivalent \*/  el.classList.add("someClass");    //----Removing a class-----    /\* jQuery \*/  $(el).removeClass("someClass");    /\* native equivalent \*/  el.classList.remove("someClass");    //----Does it have class---    /\* jQuery \*/  if($(el).hasClass("someClass"))    /\* native equivalent \*/  if(el.classList.contains("someClass")) |

当我们简单地逐个设置Css的属性，而并非将它们全部传递给JQuery的Css函数时，性能明显会快很多。而且，真的不会添加什么额外的代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // get reference to a DOM element  var el = document.querySelector(".main-content");    //----Setting multiple CSS properties----    /\* jQuery \*/  $(el).css({    background: "#FF0000",    "box-shadow": "1px 1px 5px 5px red",    width: "100px",    height: "100px",    display: "block"  });    /\* native equivalent \*/  el.style.background = "#FF0000";  el.style.width = "100px";  el.style.height = "100px";  el.style.display = "block";  el.style.boxShadow = "1px 1px 5px 5px red"; |

**jQuery show 与 hide 的原生实现**

show()与hide()应该也是JQuery中十分常用的方法，原生JS实现同样轻松。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // get reference to a DOM element  var el = document.querySelector(".main-content");    //----show() or hide()----    /\* jQuery \*/  $(el).show();  $(el).hide();    /\* native equivalent \*/  el.style.display = '';  el.style.display = 'none'; |

**jQuery 事件绑定的原生实现**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // get reference to a DOM element  var el = document.querySelector(".main-content");    //----Event Listener off----    /\* jQuery \*/  $(el).off(eventName, eventHandler);    /\* native equivalent \*/  el.removeEventListener(eventName, eventHandler);      //----Event Listener  on----    /\* jQuery \*/  $(el).on(eventName, eventHandler);    /\* native equivalent \*/  el.addEventListener(eventName, eventHandler); |

文中出现了诸如querySelector、classList等方法，确实需要考虑兼容问题，关于它们的兼容性，请猛戳这里：[*querySelector/querySelectorAll*](http://caniuse.com/#search=queryselector)*,*[*classList*](http://caniuse.com/#search=classlist)*,*[*getElementsByClassName*](http://caniuse.com/#search=getelementsbyclassname)*,*[*createDocumentFragment*](http://www.quirksmode.org/dom/w3c_core.html#miscellaneous)*.*

最后不得不说的是 jQuery 的确是一个了不起的库。但是如果我们可以同样轻松的使用原生 JS 实现 JQuery，何乐而不为呢？也许我有点吹毛求疵，这一点性能优化可能是可有可无，只不过最近在开发 H5 游戏，你会发现当你的游戏无法顺畅地跑 60 帧（60FPS，每秒刷新60次）的时候，每一丝性能的优化带来的提升都是宝贵的。