property 和 attribute非常容易混淆，两个单词的中文翻译也都非常相近（property：属性，attribute：特性），但实际上，二者是不同的东西，属于不同的范畴。

* property是DOM中的属性，是JavaScript里的对象；
* attribute是HTML标签上的特性，它的值只能够是字符串；

基于JavaScript分析property 和 attribute

html中有这样一段代码：

<input id="in\_1" value="1" sth="whatever">

简单的在html页面上创建一个input输入栏（注意在这个标签中添加了一个DOM中不存在的属性“sth”），此时在JS执行如下语句

var in1 = document.getElementById('in\_1');

执行语句

console.log(in1);

从console的打印结果，可以看到in1含有一个名为“attributes”的属性，它的类型是NamedNodeMap，同时还有“id”和“value”两个基本的属性，但没有“sth”这个自定义的属性。

attributes: NamedNodeMap

value: "1"

id: "in\_1"

有些console可能不会打印in1上的属性，那么可以执行以下命令打印要观察的属性：

console.log(in1.id); // 'in\_1'

console.log(in1.value); // 1

console.log(in1.sth); // undefined

可以发现，标签中的三个属性，只有“id”和“value”会在in1上创建，而“sth”不会被创建。这是由于，每一个DOM对象都会有它默认的基本属性，而在创建的时候，它只会创建这些基本属性，我们在TAG标签中自定义的属性是不会直接放到DOM中的。

我们做一个额外的测试，创建另一个input标签，并执行类似的操作：

html：

<input id="in\_2">

JS：

var in2 = document.getElementById('in\_2');

console.log(in2);

从打印信息中可以看到：

id: "in\_2"

value: null

尽管我们没有在TAG中定义“value”，但由于它是DOM默认的基本属性，在DOM初始化的时候它照样会被创建。由此我们可以得出结论：

* DOM有其默认的基本属性，而这些属性就是所谓的“property”，无论如何，它们都会在初始化的时候再DOM对象上创建。
* 如果在TAG对这些属性进行赋值，那么这些值就会作为初始值赋给DOM的同名property。

现在回到第一个input（“#in\_1”），我们就会问，“sth”去哪里了？别急，我们把attributes这个属性打印出来看看

console.log(in2);

上面有几个属性：

0: id

1: value

2: sth

length: 3

\_\_proto\_\_: NamedNodeMap

原来“sth”被放到了attributes这个对象里面，这个对象按顺序记录了我们在TAG中定义的属性和属性的数量。此时，如果再将第二个input标签的attributes打印出来，就会发现只有一个“id”属性，“length”为1。

从这里就可以看出，attributes是属于property的一个子集，它保存了HTML标签上定义属性。如果再进一步探索attitudes 中的每一个属性，会发现它们并不是简单的对象，它是一个Attr类型的对象，拥有NodeType、NodeName等属性。关于这一点，稍后再研究。注意，打印attribute属性不会直接得到对象的值，而是获取一个包含属性名和值的字符串，如：

console.log(in1.attibutes.sth); // 'sth="whatever"'

由此可以得出：

* HTML标签中定义的属性和值会保存该DOM对象的attributes属性里面；
* 这些attribute属性的JavaScript中的类型是Attr，而不仅仅是保存属性名和值这么简单；

那么，如果我们更改property和attribute的值会出现什么效果呢？执行如下语句：

in1.value = 'new value of prop';

console.log(in1.value); // 'new value of prop'

console.log(in1.attributes.value); // 'value="1"'

此时，页面中的输入栏的值变成了“new value of prop”，而propety中的value也变成了新的值，但attributes却仍然是“1”。从这里可以推断，property和attribute的同名属性的值并不是双向绑定的。

如果反过来，设置attitudes中的值，效果会怎样呢？

in1.attributes.value.value = 'new value of attr';

console.log(in1.value); // 'new value of attr'

console.log(in1.attributes.value); // 'new value of attr'

此时，页面中的输入栏得到更新，property中的value也发生了变化。此外，执行下面语句也会得到一样的结果

in1.attributes.value.nodeValue = 'new value of attr';

由此，可得出结论：

* property能够从attribute中得到同步；
* attribute不会同步property上的值；
* attribute和property之间的数据绑定是单向的，attribute->property；
* 更改property和attribute上的任意值，都会将更新反映到HTML页面中；

基于jQuery分析attribute和property

那么jQuery中的attr和prop方法是怎样的呢？

首先利用jQuery.prop来测试

$(in1).prop('value', 'new prop form $');

console.log(in1.value); // 'new prop form $'

console.log(in1.attributes.value); // '1'

输入栏的值更新了，但attribute并未更新。

然后用jQuery.attr来测试

$(in1).attr('value', 'new attr form $');

console.log(in1.value); // 'new attr form $'

console.log(in1.attributes.value); // 'new attr form $'

输入栏的值更新了，同时property和attribute都更新了。

从上述测试的现象可以推断，jQuery.attr和jQuery.prop基本和原生的操作方法效果一致，property会从attribute中获取同步，然而attribute不会从property中获取同步。那么jQuery到底是如何实现的呢？

下面，我们来看看jQuery.attr和jQuery.prop的源码。

jQuery源码

$().prop源码

jQuery.fn.extend({

prop: function( name, value ) {

return access( this, jQuery.prop, name, value, arguments.length > 1 );

},

... // removeProp方法

});

$().attr源码

jQuery.fn.extend({

attr: function( name, value ) {

return access( this, jQuery.attr, name, value, arguments.length > 1 );

},

... // removeAttr方法

});

无论是attr还是prop，都会调用access方法来对DOM对象的元素进行访问，因此要研究出更多内容，就必须去阅读access的实现源码。

jQuery.access

// 这是一个多功能的函数，能够用来获取或设置一个集合的值

// 如果这个“值”是一个函数，那么这个函数会被执行

// @param elems, 元素集合

// @param fn, 对元素进行处理的方法

// @param key, 元素名

// @param value, 新的值

// @param chainable, 是否进行链式调用

// @param emptyGet,

// @param raw, 元素是否一个非function对象

var access = jQuery.access = function( elems, fn, key, value, chainable, emptyGet, raw ) {

var i = 0, // 迭代计数

length = elems.length, // 元素长度

bulk = key == null; // 判断是否有特定的键（属性名）

// 如果存在多个属性，递归调用来逐个访问这些值

if ( jQuery.type( key ) === "object" ) {

chainable = true;

for ( i in key ) {

jQuery.access( elems, fn, i, key[i], true, emptyGet, raw );

}

// 设置一个值

} else if ( value !== undefined ) {

chainable = true;

if ( !jQuery.isFunction( value ) ) { // 如果值不是一个function

raw = true;

}

if ( bulk ) {

// Bulk operations run against the entire set

// 如果属性名为空且属性名不是一个function，则利用外部处理方法fn和value来执行操作

if ( raw ) {

fn.call( elems, value );

fn = null;

// ...except when executing function values

// 如果value是一个function,那么就重新构造处理方法fn

// 这个新的fn会将value function作为回调函数传递给到老的处理方法

} else {

bulk = fn;

fn = function( elem, key, value ) {

return bulk.call( jQuery( elem ), value );

};

}

}

if ( fn ) { // 利用处理方法fn对元素集合中每个元素进行处理

for ( ; i < length; i++ ) {

fn( elems[i], key, raw ? value : value.call( elems[i], i, fn( elems[i], key ) ) );

// 如果value是一个funciton,那么首先利用这个函数返回一个值并传入fn

}

}

}

return chainable ?

elems : // 如果是链式调用,就返回元素集合

// Gets

bulk ?

fn.call( elems ) :

length ? fn( elems[0], key ) : emptyGet;

};

access方法虽然不长，但是非常绕,要完全读懂并不简单，因此可以针对jQuery.fn.attr的调用来简化access。

jQuery.fn.attr/ jQuery.fn.prop 中的access调用

$().attr的调用方式：

* $().attr( propertyName ) // 获取单个属性
* $().attr( propertyName, value ) // 设置单个属性
* $().attr( properties ) // 设置多个属性
* $().attr( propertyName, function ) // 对属性调用回调函数

prop的调用方式与attr是一样的，在此就不重复列举。为了简单起见，在这里只对第一和第二种调用方式进行研究。

调用语句：

access( this, jQuery.attr, name, value, arguments.length > 1 )；

简化的access：

// elems 当前的jQuery对象,可能包含多个DOM对象

// fn jQuery.attr方法

// name 属性名

// value 属性的值

// chainable 如果value为空,则chainable为false,否则chainable为true

var access = jQuery.access = function( elems, fn, key, value, chainable, emptyGet, raw ) {

var i = 0, // 迭代计数

length = elems.length, // 属性数量

bulk = false； // key != null

if ( value !== undefined ) { // 如果value不为空,则为设置新值,否则返回该属性的值

chainable = true;

raw = true; // value不是function

if ( fn ) { // fn为jQuery.attr

for ( ; i < length; i++ ) {

fn( elems[i], key, value); // jQuery.attr(elems, key, value);

}

}

}

if(chainable) { // value不为空,表示是get

return elems； // 返回元素实现链式调用

} else {

if(length) { // 如果元素集合长度不为零，则返回第一个元素的属性值

return fn(elems[0], key); // jQuery.attr(elems[0], key);

} else {

return emptyGet； // 返回一个默认值，在这里是undefined

}

}

};

通过简化代码，可以知道，access的作用就是遍历上一个$调用得到的元素集合，对其调用fn函数。在jQuery.attr和 jQuery.prop里面，就是利用access来遍历元素集合并对其实现对attribute和property的控制。access的源码里面有多 段条件转移代码，看起来眼花缭乱，其最终目的就是能够实现对元素集合的变量并完成不同的操作，复杂的代码让jQuery的接口变得更加简单，能极大提高代 码重用性，意味着减少了代码量，提高代码的密度从而使JS文件大小得到减少。

这些都是题外话了，现在回到$().attr和$().prop的实现。总的说,这两个原型方法都利用access对元素集进行变量，并对每个元素 调用jQuery.prop和jQuery.attr方法。要注意，这里的jQuery.prop和jQuery.attr并不是原型链上的方法，而是 jQuery这个对象本身的方法，它是使用jQuery.extend进行方法扩展的（jQuery.fn.prop和jQuery.fn.attr是使 用jQuery.fn.extend进行方法扩展的）。

下面看看这两个方法的源码。

jQury.attr

jQuery.extend({

attr: function( elem, name, value ) {

var hooks, ret,

nType = elem.nodeType; // 获取Node类型

// 如果 elem是空或者NodeType是以下类型

// 2: Attr, 属性, 子节点有Text, EntityReference

// 3: Text, 元素或属性中的文本内容

// 8: Comment, 注释

// 不执行任何操作

if ( !elem || nType === 3 || nType === 8 || nType === 2 ) {

return;

}

// 如果支持attitude方法, 则调用property方法

if ( typeof elem.getAttribute === strundefined ) {

return jQuery.prop( elem, name, value );

}

// 如果elem的Node类型不是元素(1)

if ( nType !== 1 || !jQuery.isXMLDoc( elem ) ) {

name = name.toLowerCase();

// 针对浏览器的兼容性，获取钩子函数，处理一些特殊的元素

hooks = jQuery.attrHooks[ name ] ||

( jQuery.expr.match.bool.test( name ) ? boolHook : nodeHook );

}

if ( value !== undefined ) { // 如果value不为undefined，执行"SET"

if ( value === null ) { // 如果value为null，则移除attribute

jQuery.removeAttr( elem, name );

} else if ( hooks && "set" in hooks && (ret = hooks.set( elem, value, name )) !== undefined ) {

return ret; // 使用钩子函数

} else { // 使用Dom的setAttribute方法

elem.setAttribute( name, value + "" ); // 注意，要将value转换为string，因为所有attribute的值都是string

return value;

}

// 如果value为undefined，就执行"GET"

} else if ( hooks && "get" in hooks && (ret = hooks.get( elem, name )) !== null ) {

return ret; // 使用钩子函数

} else {

ret = jQuery.find.attr( elem, name ); // 实际上调用了Sizzle.attr，这个方法中针对兼容性问题作出处理来获取attribute的值

// 返回获得的值

return ret == null ?

undefined :

ret;

}

},

...

});

从代码可以发现，jQuery.attr调用的是getAttribute和setAttribute方法。

jQeury.prop

jQuery.extend({

...

prop: function( elem, name, value ) {

var ret, hooks, notxml,

nType = elem.nodeType;

// 过滤注释、Attr、元素文本内容

if ( !elem || nType === 3 || nType === 8 || nType === 2 ) {

return;

}

notxml = nType !== 1 || !jQuery.isXMLDoc( elem );

if ( notxml ) { // 如果不是元素

name = jQuery.propFix[ name ] || name; // 修正属性名

hooks = jQuery.propHooks[ name ]; // 获取钩子函数

}

if ( value !== undefined ) { // 执行"SET"

return hooks && "set" in hooks && (ret = hooks.set( elem, value, name )) !== undefined ?

ret : // 调用钩子函数

( elem[ name ] = value ); // 直接对elem[name]赋值

} else { // 执行"GET"

return hooks && "get" in hooks && (ret = hooks.get( elem, name )) !== null ?

ret : // 调用钩子函数

elem[ name ]; // 直接返回elem[name]

}

},

...

});

jQuery.prop则是直接对DOM对象上的property进行操作。

通过对比jQuery.prop和jQuery.attr可以发现，前者直接对DOM对象的property进行操作，而后者会调用 setAttribute和getAttribute方法。setAttribute和getAttribute方法又是什么方法呢？有什么效果？

setAttribute和getAttribute

基于之前测试使用的输入框，执行如下代码：

in1.setAttribute('value', 'new attr from setAttribute');

console.log(in1.getAttribute('value')); // 'new attr from setAttribute'

console.log(in1.value); // 'new attr from setAttribute'

console.log(in1.attributes.value); // 'value=​"new attr from setAttribute"',实际是一个Attr对象

执行完setAttribute以后，就如同直接更改attributes中的同名属性；  
而getAttribute的结果与访问property的结果一模一样，而不会像直接访问attritudes那样返回一个Attr对象。

特殊的例子

href

然而，是不是所有标签，所有属性都维持保持这样的特性呢？下面我们看看href这个属性/特性。

首先在html中创建一个标签：

<a href='page\_1.html' id='a\_1'></a>

在JS脚本中执行如下代码：

console.log(a1.href); // 'file:///D:/GitHub/JS/html/test\_01/page\_1.html'

console.log(a1.getAttribute('href')); // 'page\_1.html'

可以看到，property中保存的是绝对路径，而attribute中保存的是相对路径。那么，如果更改了这些值会发生什么情况呢？

更改attribute：

a1.setAttribute('href', 'page\_2.html'); // 相对路径

console.log(a1.href); // 'file:///D:/GitHub/JS/html/test\_01/page\_2.html'

console.log(a1.getAttribute('href')); // 'page\_2.html'

a1.setAttribute('href', '/page\_3.html'); // 根目录路径

console.log(a1.href); // 'file:///D:/page\_3.html'

console.log(a1.getAttribute('href')); // '/page\_3.html'

更改property：

a1.href = 'home.html'; // 相对路径

console.log(a1.href); // 'file:///D:/GitHub/JS/html/test\_01/home.html'

console.log(a1.getAttribute('href')); // 'home.html'

a1.href = '/home.html'; // 根目录路径

console.log(a1.href); // 'file:///D:/home.html'

console.log(a1.getAttribute('href')); // '/home.html'

从这里可以发现，href是特殊的属性/特性，二者是双向绑定的，更改任意一方，都会导致另一方的的值发生改变。而且，这并不是简单的双向绑定，property中的href永远保存绝对路径，而attribute中的href则是保存相对路径。

看到这里，attribute和property的区别又多了一点，然而，这又让人变得更加疑惑了。是否还有其他类似的特殊例子呢？

id

尝试改变property中的id：

a1.id = 'new\_id';

console.log(a1.id); // 'new\_id'

console.log(a1.getAttribute('id')); // 'new\_id'

天呀,现在attribute中的id从property中的id发生了同步，数据方向变成了property <=> attribute；

disabled

再来看看disabled这个属性，我们往第一个添加“disabled”特性：

<input id="in\_1" value="1" sth="whatever" disabled='disabled'> // 此时input已经被禁用了

然后执行下面的代码：

console.log(in1.disabled); // true

in1.setAttribute('disabled', false); // 设置attribute中的disabled，无论是false还是null都不会取消禁用

console.log(in1); // true

console.log(in1.getAttribute('disabled')); // 'false'

改变attributes中的disabled不会改变更改property，也不会取消输入栏的禁用效果。  
如果改成下面的代码：

console.log(in1.disabled); // true

in1.disabled = false; // 取消禁用

console.log(in1.disabled); // false

console.log(in1.getAttribute('disabled')); // null，attribute中的disabled已经被移除了

又或者：

console.log(in1.disabled); // true

in1.removeAttribute（'disabled'); // 移除attribute上的disabled来取消禁用

console.log(in1.disabled); // false

console.log(in1.getAttribute('disabled')); // null，attribute中的disabled已经被移除了

可以发现，将property中的disabled设置为false，会移除attributes中的disabled。这样数据绑定又变成了，property<=>attribute;

所以property和attritude之间的数据绑定问题并不能单纯地以“property<-attribute”来说明。

总结

分析了这么多，对property和attribute的区别理解也更深了，在这里总结一下：

创建

* DOM对象初始化时会在创建默认的基本property；
* 只有在HTML标签中定义的attribute才会被保存在property的attributes属性中；
* attribute会初始化property中的同名属性，但自定义的attribute不会出现在property中；
* attribute的值都是字符串；

数据绑定

* attributes的数据会同步到property上，然而property的更改不会改变attribute；
* 对于value，class这样的属性/特性，数据绑定的方向是单向的，attribute->property；
* 对于id而言，数据绑定是双向的，attribute<=>property；
* 对于disabled而言，property上的disabled为false时，attribute上的disabled必定会并存在，此时数据绑定可以认为是双向的；

使用

* 可以使用DOM的setAttribute方法来同时更改attribute；
* 直接访问attributes上的值会得到一个Attr对象，而通过getAttribute方法访问则会直接得到attribute的值；
* 大多数情况（除非有浏览器兼容性问题），jQuery.attr是通过setAttribute实现，而jQuery.prop则会直接访问DOM对象的property；

到这里为止，得出，property是DOM对象自身就拥有的属性，而attribute是我们通过设置HTML标签而给之赋予的特性，attribute和property的同名属性/特性之间会产生一些特殊的数据联系，而这些联系会针对不同的属性/特性有不同的区别。

事实上，在这里，property和attribute之间的区别和联系难以用简单的技术特性来描述，我在StackFlow上找到如下的回答，或者会更加接近于真正的答案：

These words existed way before Computer Science came around.

Attribute is a quality or object that we attribute to someone or something. For example, the scepter is an attribute of power and statehood.

Property is a quality that exists without any attribution. For example, clay has adhesive qualities; or, one of the properties of metals is electrical conductivity. Properties demonstrate themselves though physical phenomena without the need attribute them to someone or something. By the same token, saying that someone has masculine attributes is self-evident. In effect, you could say that a property is owned by someone or something.

To be fair though, in Computer Science these two words, at least for the most part, can be used interchangeably - but then again programmers usually don't hold degrees in English Literature and do not write or care much about grammar books :).

最关键的两句话：

* attribute（特性），是我们赋予某个事物的特质或对象。
* property（属性），是早已存在的不需要外界赋予的特质。