

# MutationObserver

`MutationObserver` 给开发者们提供了一种能在某个范围内的 `DOM` 树发生变化时作出适当反应的能力. 该API 设计用来替换掉在 DOM3 事件规范中引入的 `Mutation` 事件.

## 构造函数

### `MutationObserver( )`

该构造函数用来实例化一个新的 `Mutation` 观察者对象.

```
1 MutationObserver(  
2     function callback  
3 );
```

参数

**callback**

该回调函数会在指定的 DOM 节点(目标节点)发生变化时被调用. 在调用时, 观察者对象会传给该函数两个参数, 第一个参数是个包含了若干个 `MutationRecord` 对象的数组, 第二个参数则是这个观察者对象本身.

## 实例方法

```
void observe( Node target, optional MutationObserverInit options );
```

```
void disconnect();
```

```
Array takeRecords();
```

### `observe( )`

给当前观察者对象注册需要观察的目标节点, 在目标节点(还可以同时观察其后代节点)发生 DOM 变化时收到通知.

```
void observe(  
    Node target,  
    optional MutationObserverInit options  
);
```


参数

target

观察该节点是否会发生DOM变化.

options

一个MutationObserverInit对象,指定要观察的DOM变化类型.

 注: 向一个元素添加 observer 和 addEventListener 类似, 注册多次不会有任何影响. 即是说, 如果你注册了两次, 回调函数不会被调用两次, 你也不必执行两次 disconnect() 以停止观察. 换句话说, 一旦某个元素被注册观察后, 使用相同的 observer 实例再次注册不会发生任何变化. 当然, 如果回调对象不同, 那么他会向这个元素添加另一个观察者.

## disconnect()

让该观察者对象停止观察指定目标的DOM变化.直到再次调用其observe()方法,该观察者对象包含的回调函数都不会再被调用.

```
1 | void disconnect();
```

## takeRecords()

清空观察者对象的记录队列,并返回里面的内容.

```
1 | Array takeRecords();
```

返回值

返回一个包含了MutationRecords对象的数组.

# MutationObserverInit

MutationObserverInit是一个用来配置观察者对象行为的对象,该对象可以拥有下面这些属性:

 注: childList, attributes, 或者characterData三个属性中必须至少有一个为true.否则,会抛出异常"An invalid or illegal string was specified".

属性	描述
childList	如果需要观察目标节点的子节点(新增了某个子节点,或者移除了某个子节点),则设置为true.
attributes	如果需要观察目标节点的属性节点(新增或删除了某个属性,以及某个属性的属性值发生了变化),则设置为true.
	如果目标节点为characterData节点(一种抽象接口,具体可以为文本节

2016/9/18MutationObserver - Web API 接口 | MDN

characterData	点,注释节点,以及处理指令节点)时,也要观察该节点的文本内容是否发生变化,则设置为true.
subtree	除了目标节点,如果还需要观察目标节点的所有后代节点(观察目标节点所包含的整棵DOM树上的上述三种节点变化),则设置为true.
attributeOldValue	在attributes属性已经设为true的前提下,如果需要将发生变化的属性节点之前的属性值记录下来(记录到下面MutationRecord对象的oldValue属性中),则设置为true.
characterDataOldValue	在characterData属性已经设为true的前提下,如果需要将发生变化的characterData节点之前的文本内容记录下来(记录到下面MutationRecord对象的oldValue属性中),则设置为true.
attributeFilter	一个属性名数组(不需要指定命名空间),只有该数组中包含的属性名发生变化时才会被观察到,其他名称的属性发生变化后会被忽略.

## MutationRecord

MutationRecord对象会作为第一个参数传递给观察者对象包含的回调函数,该对象有下面这些属性:

属性	类型	描述
type	String	如果是属性发生变化,则返回attributes.如果是一个CharacterData节点发生变化,则返回characterData,如果是目标节点的某个子节点发生了变化,则返回childList.
target	Node	返回此次变化影响到的节点,具体返回那种节点类型是根据type值的不同而不同的. 如果type为attributes,则返回发生变化的属性节点所在的元素节点,如果type值为characterData,则返回发生变化的这个characterData节点.如果type为childList,则返回发生变化的子节点的父节点.
addedNodes	NodeList	返回被添加的节点,或者为null.
removedNodes	NodeList	返回被删除的节点,或者为null.
previousSibling	Node	返回被添加或被删除的节点的前一个兄弟节点,或者为null.
nextSibling	Node	返回被添加或被删除的节点的后一个兄弟节点,或者为null.
attributeName	String	返回变更属性的本地名称,或者为null.
attributeNamespace	String	返回变更属性的命名空间,或者为null.
		根据type值的不同,返回的值也会不同.如果type为attributes,则返回该属性变化之前的属性值.如果type

oldValue	String	为characterData,则返回该节点变化之前的文本数据.如果type为childList,则返回null.
----------	--------	--

## 例子

下面的例子来自 [这篇博文](#).

```
1 // Firefox和Chrome早期版本中带有前缀
2 var MutationObserver = window.MutationObserver || window.WebKitMutationOb
3
4 // 选择目标节点
5 var target = document.querySelector('#some-id');
6
7 // 创建观察者对象
8 var observer = new MutationObserver(function(mutations) {
9     mutations.forEach(function(mutation) {
10         console.log(mutation.type);
11     });
12 });
13
14 // 配置观察选项:
15 var config = { attributes: true, childList: true, characterData: true }
16
17 // 传入目标节点和观察选项
18 observer.observe(target, config);
19
20 // 随后,你还可以停止观察
21 observer.disconnect();
```

假设target为当前文档中某个已知的节点,observer为某个已经实例化的MutationObserver观察者对象,则:

```
observer.observe(target, {childList:true}) //childList: true
target.appendChild(document.createElement("div")) //添加了
target.appendChild(document.createTextNode("foo")) //添加了
target.removeChild(target.childNodes[0]) //移除第
target.childNodes[0].appendChild(document.createElement("div")) //为第一
```

```
observer.observe(target, {childList:true,subtree:true}) //subtree: true
observer.observe(document, {childList:true,subtree:true}) //如果target是document
observer.observe(document, {childList:true,attributes:true,characterData:true,subtree:true})
```

```

observer.observe(target, {childList:true})           //假设此
target.childNodes[0].data = "bar"                   //不会触
observer.observe(target, {childList:true,characterData:true}) //加上c
target.childNodes[0].data = "bar"                   //还是不
observer.observe(target, {childList:true,characterData:true,subtree:true}) //;
target.childNodes[0].data = "bar"                   //触发了

```

```

observer.observe(target, {attributes:true})           //只观察
target.setAttribute("foo","bar")                     //不管f
target.setAttribute("foo","bar")                     //即使前
target.removeAttribute("foo")                       //移除f
target.removeAttribute("foo")                       //不会触
observer.observe(target, {attributes:true,attributeFilter:["bar"]}) //指定要
target.setAttribute("foo","bar")                     //不会触
target.setAttribute("bar","foo")                     //触发了

```

## 外部链接

- [A brief overview](#)
- [A more in-depth discussion](#)
- [A screencast by Chromium developer Rafael Weinstein](#)
- [The mutation summary library](#)
- [The DOM4 specification](#) which defines the `MutationObserver` interface

## 浏览器兼容性

Desktop

Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	18  <b>webkit</b> 26	14 (14)	11	15	6.0  <b>WebKit</b>