# **js-cookie使用方法**

**js-cookie用来处理cookie相关的插件**

## **1、项目中引用：**

npm install --save js-cookie

## **2、js-cookie的使用：**

安装好js-cookie插件后，在我们需要处理cookie的地方，简单的通过import引入就可以使用了

import Cookies from 'js-cookie'

**3、直接引用cdn：**

<script src="https://cdn.jsdelivr.net/npm/js-cookie@2/src/js.cookie.min.js"></script>

## **js-cookie的増、查、删**

### **添加cookie**

*// 创建一个名称为name，对应值为value的cookie，由于没有设置失效时间，默认失效时间为该网站关闭时*

Cookies.set(name, value)

*// 创建一个有效时间为7天的cookie*

Cookies.set(name, value, { expires: 7 })

*// 创建一个带有路径的cookie*

Cookies.set(name, value, { path: '' })

*// 创建一个value为对象的cookie*const obj = { name: 'ryan' }

Cookies.set('user', obj) //即设置了一个json，获取的时候，需要使用JSON.parse()  
Cookies.set('name', 'value', { expires: 7, path: '' }); //7天过期

Cookies.set('name', { foo: 'bar' }); //设置一个json

需要注意的是，通过Cookies.set(name, value)添加cookie时，即使添加时的value值类型为number，添加后获取到的value值的类型会被转换成string类型。

cookie添加后，所有的请求接口都会自动带上cookie值，如果没有设置cookie的失效时间，默认就是该网站关闭时cookie失效。

### **获取cookie**

*// 获取指定名称的cookie*

Cookies.get(name) *// value*

*// 获取value为对象的cookie*const obj = { name: 'ryan' }

Cookies.set('user', obj)JSON.parse(Cookies.get('user'))

*// 获取所有cookie*

Cookies.get()

获取cookie时，如果cookie中不存在该名称对应的记录，则会返回undefined。当value为对象时，获取的cookie需要通过JSON.parse()解析

### **删除cookie**

*// 删除指定名称的cookie*

Cookies.remove(name) *// value*

*// 删除带有路径的cookie*

Cookies.set(name, value, { path: '' })

Cookies.remove(name, { path: '' })

删除带有路径path的cookie时，不能通过简单的Cookies.remove(name)进行删除，需要带上路径

## **命名空间**

如果担心不小心修改掉Cookies中的数据，可以用noConflict方法定义一个新的cookie。

var Cookies2 = Cookies.noConflict();

Cookies2.set('name', 'value');

## **json相关**

js-cookie允许你向cookie中存储json信息。

如果你通过set方法，传入Array或类似对象，而不是简单的string，那么js-cookie会将你传入的数据用JSON.stringify转换为string保存。

Cookies.set('name', { foo: 'bar' });

Cookies.get('name'); *// => '{"foo":"bar"}'*

Cookies.get(); *// => { name: '{"foo":"bar"}' }*

如果你用getJSON方法获取cookie，那么js-cookie会用JSON.parse解析string并返回。

Cookies.getJSON('name'); *// => { foo: 'bar' }*

Cookies.getJSON(); *// => { name: { foo: 'bar' } }*

## **set方法支持的属性**

expires:  
 定义有效期。如果传入Number，那么单位为天，你也可以传入一个Date对象，表示有效期至Date指定时间。默认情况下cookie有效期截止至用户退出浏览器。

path:  
 string，表示此cookie对哪个地址可见。默认为”/”。

domain:  
 string，表示此cookie对哪个域名可见。设置后cookie会对所有子域名可见。默认为对创建此cookie的域名和子域名可见。

secure:  
 true或false，表示cookie传输是否仅支持https。默认为不要求协议必须为https。

**HttpOnly属性**

　　如果在Cookie中设置了"HttpOnly"属性，那么通过程序(JS脚本、Applet等)将无法读取到Cookie信息，这样能有效的防止XSS攻击。

　　即使应答头中含有HttpOnly属性，当用户浏览有效域中的站点时，这个cookie仍会被自动发送。但是，却不能够在Internet Explorer 6 SP1中使用脚本来访问该cookie，即使起初建立该cookie的那个Web 站点也不例外。

HttpOnly是包含在Set-Cookie HTTP响应头文件中的附加标志。生成cookie时使用HttpOnly标志有助于降低客户端脚本访问受保护cookie的风险（如果浏览器支持）。

这个意思就是说，如果某一个Cookie 选项被设置成 HttpOnly = true 的话，那此Cookie 只能通过服务器端修改，Js 是操作不了的，对于 document.cookie 来说是透明的。

# [js 如何判断浏览器是否禁用cookie](https://www.cnblogs.com/wangkang0320/p/6812802.html)

**navigator.cookieEnabled；**

如果浏览器启用了cookie，该属性值为true。如果禁用了cookie，则值为false。

****navigator：**** JavaScript中的一个独立的对象，他用于提供用户所使用的浏览器以及操作系统等信息，以navigator对象属性的形式来提供。

