JavaScript Cookies

Summary: in this tutorial, you'll learn about HTTP cookies and how to use JavaScript to manage the cookies effectively.

What is a cookie

An HTTP cookie is a piece of data that a server sends to a web browser. Then, the web browser stores the HTTP cookie on the user's computer and sends it back to the same server in the later requests.

An HTTP cookie is also known as a web cookie or browser cookie. And it is commonly called a cookie.

For example, the header of an HTTP response may look like this:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie:username=admin
...
```

The HTTP response sets a cookie with the name of `"username"` and value of `"admin"`. The server encodes both name and value when sending the cookie to the web browser.

The web browser stores this information and sends it back to the server via the Cookie HTTP header for the next request as follows:

```
GET /index.html HTTP/1.1
Cookie: username=admin
...
```

Why cookies

It's important to understand that **HTTP is stateless**. When you send two subsequent HTTP requests to the server, there is no link between them. In other words, the server cannot know if the two requests are from the same web browser.

Therefore, a cookie is used to tell if the two requests came from the same web browser.

In practice, cookies serve the following purposes:

- Session management – cookies allow you to manage any information that the server should remember. For example, logins, shopping carts, etc.
- Personalization – cookies allow you to store user preferences, themes, and setting specific to a user.
- Tracking – cookies help record and analyze user behaviors in advertising.

Cookie details

A cookie consists of the following information stored by the web browser:

- Name – a unique name that identifies the cookie. The cookie names are case-insensitive. It means that `Username` and `username` are the same cookies.
- Value – string value of the cookie. It must be URL-encoded.
- Domain – a domain for which the cookie is valid.
- Path – path without the domain for which the cookie should be sent to the server. For example, you can specify that the cookie is accessible only from the `https://www.javascripttutorial.net/dom/` so pages at `https://www.javascripttutoiral.net/` won't send the cookie information.
- Expiration – timestamp that indicates when the web browser should delete the cookie (or when the browser should stop sending the cookie to the server). The expiration date is set as a date in GMT format: `Wdy, DD-Mon-YYYY HH:MM:SS GMT`. The expiration date allows the cookies to be stored in the user's web browsers even after users close the web browsers.
- Secure flag – if specified, the web browser only sends the cookie to the server only via an SSL connection (`https` , not `http`)

The name, value, domain, path, expiration, and secure flag are separated by a semicolon and space. For example:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie:user=john
; expire=Tue, 12-December-2030 12:10:00 GMT; domain=javascripttutorial.net; path=/d
...
```

Note that the `secure` flag is the only part that is not a name-value pair.

Cookies in JavaScript

To manage cookies in JavaScript, you use the `document.cookie` property.

1) Get a cookie value

The following example returns a string of all cookies available to the page:

```
const str = document.cookie;
```

The `document.cookie` returns a series of name-value pairs separated by semicolons like this:

```
name1=value1;name2=value2;...
```

The following example shows the cookies of a webpage:

```
"_ga=GA1.2.336374160.1600215156; dwf_sg_task_completion=False; _gid=GA1.2.33408724.
```

Since all the names and values are URL-encoded, you need to use the `decodeURIComponent()` to decode them.

2) Set a cookie

To set a value for a cookie, you compose the cookie text in the following format:

```
name=value; expires=expirationTime; path=domainPath; domain=domainName; secure
```

...and append it to the cookie:

```
document.cookie = cookieText;
```

A cookie requires only name and value. For example:

```
document.cookie = "username=admin";
```

This example creates a cookie called `username` that has a value of `admin`. The web browser will send this cookie every time it makes a request to the server.

Since the cookie doesn't specify the expired time, it will be deleted when the web browser is closed.

The cookie text `"username=admin"` doesn't have any character that needs to be encoded.

However, it's a good practice to always use the `encodeURIComponent()` function when setting a cookie like this:

```
document.cookie = `${encodeURIComponent("username")}=${encodeURIComponent("admin")}
```

3) Remove a cookie

To remove a cookie, you need to set the cookie again with the same name, path, domain, and secure option. And you need to set the expiration date to some time in the past.

JavaScript Cookie class

The following Cookie class sets, gets, and removes a cookie:

```
class Cookie {
  static get(name) {
    const cookieName = `${encodeURIComponent(name)}=`;
```

```
    const cookie = document.cookie;
    let value = null;

    const startIndex = cookie.indexOf(cookieName);
    if (startIndex > -1) {
      const endIndex = cookie.indexOf(';', startIndex);
      if (endIndex == -1) {
        endIndex = cookie.length;
      }
      value = decodeURIComponent(
        cookie.substring(startIndex + name.length, endIndex)
      );
    }
    return value;
  }

  static set(name, value, expires, path, domain, secure) {
    let cookieText = `${encodeURIComponent(name)}=${encodeURIComponent(value)}`;
    if (expires instanceof Date) {
      cookieText += `; expires=${expires.toGMTString()}`;
    }

    if (path) cookieText += `; path=${path}`;
    if (domain) cookieText += `; domain=${domain}`;
    if (secure) cookieText += `; secure`;

    document.cookie = cookieText;
  }

  static remove(name, path, domain, secure) {
    Cookie.set(name, '', new Date(0), path, domain, secure);
  }
}
```

How it works.

The `Cookie` class has three static methods: `get()`, `set()`, and `remove()`.

1) The get() method

The `get()` method returns the value of a cookie with a specified name. To do so, it performs the following steps:

- First, find the occurrence of the cookie name by an equal sign in the `document.cookie` property.
- Second, if the cookie is available, it uses the `indexOf()` to find the end of the cookie, which is specified by the next semicolon (`;`) after that location. If the semicolon isn't available, this means that the cookie is the last one in the string.
- Third, decode the value of the cookie using the `decodeURIComponent()` function and return the decoded value.

2) The set() method

The `set()` method sets a cookie on the page. It accepts the arguments required to construct a cookie.

The `set()` method requires the first two arguments: name and value. The other arguments aren't mandatory.

The `set()` method composes a cookie text and sets it to the `document.cookie` property.

3) The remove() method

To remove a cookie, the `remove()` method sets the cookie again with the expiration date set to January 1, 1970. Note that the new `Date(0)` returns a date object whose date is January 1, 1970.

Using the Cookie class

The following shows how to use the Cookie class to set, get, and remove a cookie whose name is username and value is admin:

```
// set a cookie
Cookie.set('username', 'admin');
```

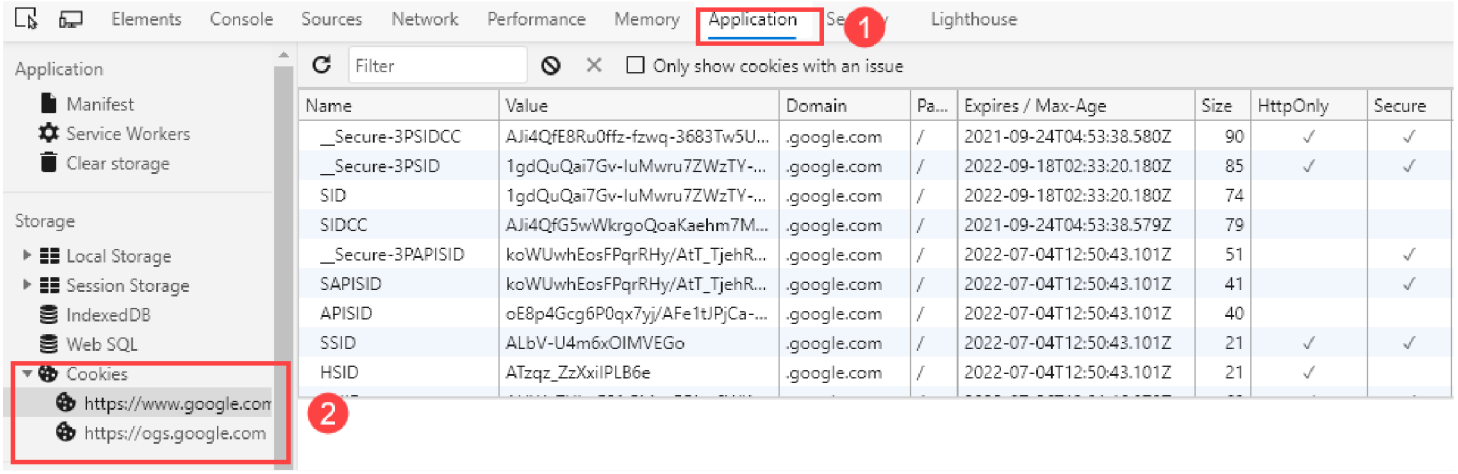
```
// get a cookie
console.log(Cookie.get('username')); // admin

// remove a cookie by a name
Cookie.remove('username');
```

View cookies with web browsers

To view the cookies on the web browser, you use the devtools.

- First, click the application tab.
- Second, select **Cookies** node under the **Storage**.

The following picture shows the cookies of Google.com:



Summary

- A cookie is a piece of data that a server sends to a web browser. The web browser then stores it in the user's computer and sends the cookie back to the same server in the subsequent requests.
- The server uses cookies for identifying if two successive requests came from the same web browser.
- To manage cookies, you use the `document.cookie` object. To make it more efficient, you can use the Cookie utility class.
- Use the `encodeURIComponent()` and `decodeURIComponent()` function to encode and decode the cookie values.