**Tìm hiểu về local storage và session storage và cookie**

**1. Local Storage**

**Giới thiệu:**

* Dữ liệu được lưu trữ vô thời hạn, nghĩa là dữ liệu sẽ không bị mất đi trừ khi bạn dùng chức năng xóa dữ liệu của trình duyệt hoặc sử sụng localStorage API để xóa. Trong trường hợp người dùng sử dụng chức năng "ẩn danh" thì nó sẽ bị xóa sau khi tab ẩn danh cuối cùng bị đóng lại.
* Khả năng lưu trữ: 5MB
* Dữ liệu không được gửi đi đến server thông qua các request header.
* Được lưu trữ trên trình duyệt của người dùng nên việc sử dụng sẽ không ảnh hưởng đến hiệu suất của trang web nhưng sẽ làm nặng trình duyệt (rất nhỏ gần như không đáng kể).
* Dữ liệu được chia sẽ giữa các tab, các cửa sổ cùng lúc nếu nó có chung nguồn gốc hay còn gọi là "same origin" (bao gồm 3 thành phần domain/port/protocol).

## Sử dụng:

Để thao tác với dữ liệu được lưu trong Local Storage, ta sử dụng 3 hàm sau đây:

### **setItem(key, value)**

Hàm này giúp lưu dữ liệu vào trong localStorage, nó nhận vào hai tham số là key và value.

localStorage.setItem("domain", "viblo.asia");

// Một số cách khác

localStorage.domain = "viblo.asia";

localStorage["domain"] = "viblo.asia";

Lưu ý rằng Local Storage chỉ cho lưu value dưới dạng string thế nên khi muốn lưu dữ liệu dưới dạng array hoặc object bạn cần phải chuyển nó sang string với hàm JSON.stringify() trước khi lưu và chuyển lại với JSON.parse() khi lấy ra.

const person = {

name: 'Nam',

job: 'student',

}

localStorage.setItem('user', JSON.stringify(person));

### **getItem(key)**

Lấy ra dữ liệu trong localStorage với key và trả về value tương ứng

const user = localStorage.getItem('user');

// Một số cách khác

const user = localStorage.user;

const user = localStorage['user'];

Nếu dùng JSON.stringify() trước khi lưu thì value lấy ra sẽ có dạng:

"{"name":"Nam","job":"student"}"

Để sử dụng, cần phải đưa nó trở lại về dạng object với JSON.parse()

const user = JSON.parse(window.localStorage.getItem('user'));

### **removeItem(key)**

Loại bỏ dữ kiệu ra khỏi localStorage với tham số key được truyền vào.

localStorage.removeItem('user');

**2. Session Storage**

**Giới thiệu:**

* Session Storage cũng giống như Local Storage như việc lưu dữ liệu được lưu trong trình duyệt của client, cùng kích thước lưu trữ, điểm khác biệt duy nhất là nó chỉ tồn tại cho đến khi người dùng đóng tab, đóng trình duyệt.
* Khả năng lưu trữ: 5MB
* Dữ liệu không được gửi đi đến server thông qua các request header.
* Dữ liệu chỉ tồn tại trong tab trình duyệt hiện tại, một tab khác dù mở cùng một trang nhưng nó sẽ có bộ nhớ khác.
* Dữ liệu vẫn tồn tại khi tải lại trang miễn là bạn chưa đóng tab đó.

## Sử dụng:

Về mặt sử dụng thì Session Storage cũng có các hàm để thao tác với dữ liệu giống hệt như Local Storage:

if (typeof(sessionStorage) !== "undefined") {

// Lưu dữ liệu

sessionStorage.setItem('domain', 'viblo.asia');

// sessionStorage.domain = 'viblo.asia';

// sessionStorage['domain'] = 'viblo.asia';

// Lấy dữ liệu

const domain = sessionStorage.getItem('domain');

// const domain = sessionStorage.domain;

// const domain = sessionStorage['domain'];

// Xóa dữ liệu

sessionStorage.removeItem('domain');

// Xóa tất cả dữ liệu

sessionStorage.clear();

// Lấy ra số lượng session storage đã lưu

const length = sessionStorage.length();

// Lấy ra tên key trong session storage với index chỉ định

const key = sessionStorage.key(0); // Key của dữ liệu được thêm gần nhất tương ứng với index = 0

} else {

console.info("Trình duyệt của bạn không hỗ trợ Session Storage");

}

**3. Cookie**

**Giới thiệu**

* Khả năng lưu trữ: thông thường là 4KB (IE là 4KB - 10KB) và khoảng vài chục cookie cho một domain (Chrome và Safari không giới hạn).
* Thông tin được gửi lên server.
* Có thể đọc ở phía máy chủ khác với Local/Session Storage chỉ đọc được ở phía máy khách.
* Có hạn sử dụng: mỗi cookie thường có một khoảng thời gian sống nhất định và tùy vào mục đích sử dụng sẽ có khoảng thời gian sử dụng khác nhau, ngoài hạn sử dụng cookie còn có thời gian sống (*max-age*).
* Có các trường như: name, value, domain, path, expire/max-age, size, secure flag, priority... thay vì 2 trường đơn thuần key, value.

## Sử dụng

Có nhiều cách để thao tác với cookie, trong bài viết này mình sẽ chỉ đề cập tới thao tác qua document.cookie trong javascript.

Để tạo ra một cookie, ta dùng cú pháp sau:

document.cookie = 'domain=viblo.asia; expires=Mon, 17 Aug 2020 7:00:00 UTC;';

Hoặc có thể thay expires bằng max-age (tính bằng giây), expries sẽ được tính lại dựa trên max-age:

document.cookie = 'domain=viblo.asia; max-age=10';

Lưu ý là định dạng của expires truyền vào phải dưới định dạng UTC String.

Để lấy ra thông tin cookie đã lưu:

let cookies = document.cookie;

Giá trị trả về sẽ là một chuỗi chứa thông tin của tất cả cookie. Ví dụ username=namdh; email=google@namdh.com; domain=viblo.asia;"

Để đơn giản hơn trong việc lấy ra giá trị của cookie, thử viết một hàm như sau:

function getCookie(name) {

let matches = document.cookie.match(new RegExp(

"(?:^|; )" + name.replace(/([\.$?\*|{}\(\)\[\]\\\/\+^])/g, '\\$1') + "=([^;]\*)"

));

return matches ? decodeURIComponent(matches[1]) : undefined;

}

Bây giờ muốn lấy ra giá trị của cookie có name là domain thì chỉ cần gọi đến hàm getCookie('domain'), giá trị trả về sẽ là value của cookie có tên đó.

Để thay đổi giá trị của cookie, đơn giản chỉ việc ghi đè giá trị mới lên cookie đã có:

document.cookie = 'domain=code.viblo.asia; expires=Mon, 17 Aug 2020 7:00:00 UTC;';

Để xóa một cookie, chỉ cần gán giá trị expries bằng một khoảng thời gian nào đó trong quá khứ hoặc đơn giản hơn là gán max-age bằng 0 hoặc một giá trị âm tùy ý.

document.cookie = 'domain=code.viblo.asia; max-age=-1;';