BACHARELADO EM SISTEMAS DE INFORMAÇÃO
CEFET-RJ NOVA FRIBURGO

Programação de Clientes Web

PROF. THIAGO DELGADO PINTO
thiago.pinto@cefet-rj.br

armazenamentos

versão: 2022.10.29

AGENDA

- armazenamentos do navegador
- sessionStorage e localStorage
    - disponibilidade
    - visão geral
    - interface Storage
    - obtendo um valor
    - guardando um valor
    - removendo um valor
    - espiando todos os valores
    - guardando e obtendo um objeto
    - evento storage [conteúdo extra]

Outros meios de armazenamento, como IndexedDB e Cache Storage, estão fora do escopo desta disciplina.

armazenamentos do navegador

Nome	Tipo	Limite padrão	Política	Bloqueante*
sessionStorage	chave-valor[1]	5 MB	LRU	Sim
localStorage	chave-valor[1]	5 MB	LRU	Sim
Cookie	chave-valor[1]	4 MB	Tempo até expiração	Sim
IndexedDB	geral	2 GB+[2]	LRU	Não
Cache API	geral	2 GB+[2]	LRU	Não
FileSystem API	arquivos	da mídia	Manual	Não

* – Bloqueia a *thread* principal.
1 – Somente *strings*.
2 – Limite pode variar conforme o motor do navegador.

Observação: *Web SQL* não foi incluído pois está obsoleto.

sessionStorage & localStorage

disponibilidade

Window API: sessionStorage

https://caniuse.com/?search=sessionStorage

Usage % of all users ?

Global 97.36%

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE
4		3.1-3.2		10.1	6-7
5-106	12-105	4-16.0	2-105	11.5-90	8-10
107	106	16.1	106	91	11
108-110		16.2-TP	107-108		

Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
	3.2-16.0	4-17.0		12-12.1		2.1-4.4.4				
106	16.1	18.0	all	64	13.4	106	105	13.1	13.18	2.5

Window API: localStorage

https://caniuse.com/?search=localStorage

Usage % of all users ?

Global 97.35%

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE
		3.1-3.2	2-3	10.1	6-7
4-106	12-105	4-16.0	3.5-105	11.5-90	8-10
107	106	16.1	106	91	11
108-110		16.2-TP	107-108		

Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
	3.2-16.0	4-17.0		12-12.1		2.1-4.4.4				
106	16.1	18.0	all	64	13.4	106	105	13.1	13.18	2.5

visão geral

ambos são usados para **armazenar dados pequenos**

usando o formato *string*

sessionStorage serve para guardar dados **temporários**

localStorage serve para guardar dados **por períodos longos**

sessionStorage

1. **escopo** dos dados é **somente** para a aba atual
   se abrir outra aba da mesma aplicação, os dados não estarão lá

2. **dados permanecem** ao **atualizar** a página (*refresh*)

3. dados são **apagados** ao **fechar a aba**

👉 use para guardar coisas temporárias, que sobrevivam a um *refresh*

localStorage

1. **escopo** dos dados é **compartilhado** entre todas as abas e janelas da mesma origem

    mesma origem = protocolo, domínio e portas iguais

2. dados *não* são apagados automaticamente

👉 use para guardar coisas por períodos mais longos

interface Storage

```
[Exposed=Window]
interface Storage {
  readonly attribute unsigned long length;
  DOMString? key(unsigned long index);
  getter DOMString? getItem(DOMString key);
  setter undefined setItem(DOMString key, DOMString value);
  deleter undefined removeItem(DOMString key);
  undefined clear();
};
```

Chave e valor são sempre do tipo `string`.

👉 Ambos `sessionStorage` e `localStorage` implementam essa interface.

https://html.spec.whatwg.org/multipage/webstorage.html#the-storage-interface

acesso

ambos são acessíveis via objeto **window**

e não são acessíveis via *Web Workers*

exemplo:

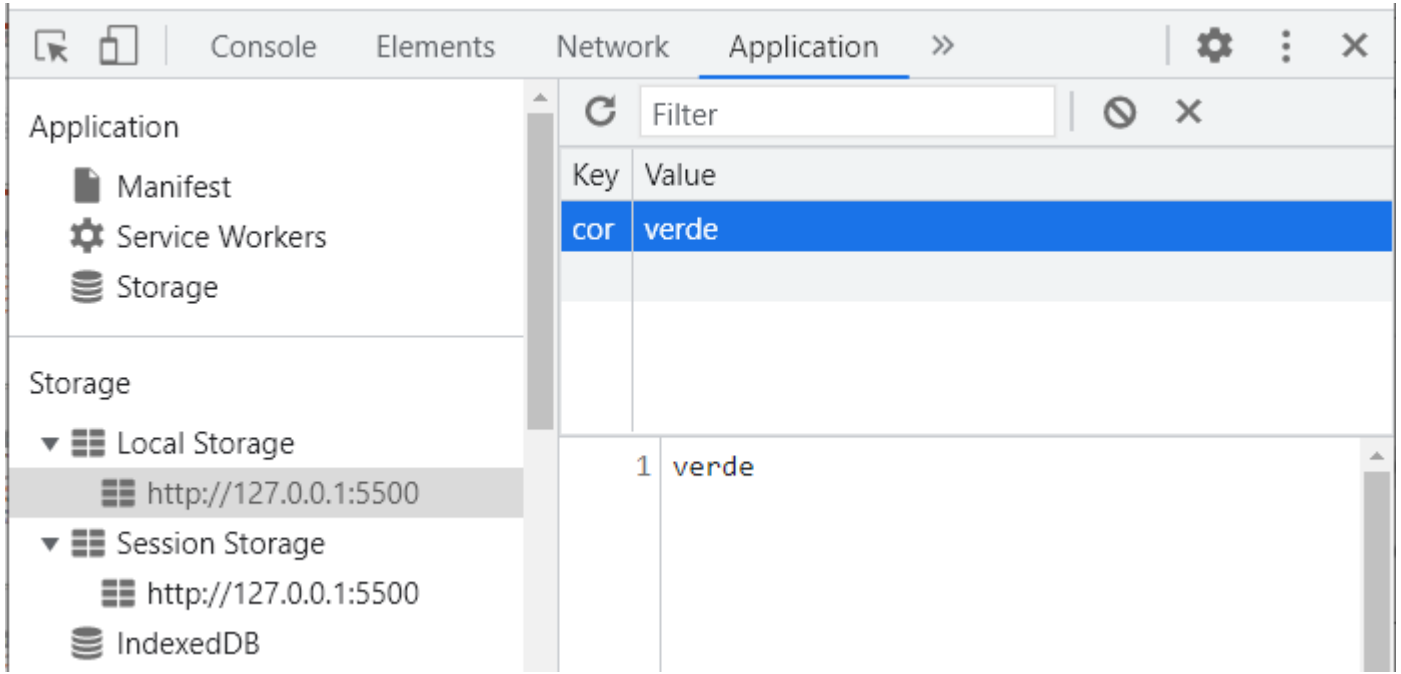
```
window.localStorage.setItem( cor', 'verde' );
window.sessionStorage.setItem( 'nome', 'Ana' );
```

o que é, no navegador, semelhante a:

```
localStorage.setItem( 'cor', 'verde' );
sessionStorage.setItem( 'nome', 'Ana' );
```

visualização no navegador

via aba *Application*



obtendo um valor

```
let nome;
// Formas equivalentes
nome = window.sessionStorage.getItem( 'nomeUsuario' );
nome = sessionStorage.getItem( 'nomeUsuario' );
nome = sessionStorage[ 'nomeUsuario' ];
nome = sessionStorage.nomeUsuario;
```

definindo um valor

```
const nome = document.getElementById( 'nome' ).value;

// Formas equivalentes
window.localStorage.setItem( 'nomeUsuario', nome );
localStorage.setItem( 'nomeUsuario', nome );
localStorage[ 'nomeUsuario' ] = nome;
localStorage.nomeUsuario = nome;
```

removendo um valor

```
// Remove um valor pela chave
window.sessionStorage.removeItem( 'nomeUsuario' );

// Remove todos os valores
window.sessionStorage.clear();
```

espiando todos os valores

```
for ( let i = 0, max = localStorage.length; i < max; i++ ) {
  const chave = localStorage.key( i );
  const valor = localStorage.getItem( chave );
  console.log( chave, '=', valor );
}
```

guardando e obtendo um objeto

```
let usuario = { nome: 'Ana', email: 'ana@exemplo.com' };

// Guarda como uma string (serializa)
sessionStorage.setItem( 'usuario', JSON.stringify( usuario ) );

// Transforma de string para objeto (deserializa)
usuario = JSON.parse( sessionStorage.getItem( 'usuario' ) );
```

evento

storage

evento `storage`

ocorre quando um dado é **inserido**, **atualizado** ou **removido**

é disparado em frames/abas/janelas diferentes da atual (!)

ou seja, só outros frames/abas/janelas vão receber o evento

para `sessionStorage` ocorre apenas entre frames da mesma página

já que abas/janelas não têm acesso às *sessionStorages* das outras

para `localStorage` ocorre entre abas/janelas da mesma origem

é disparado no objeto `window`

dados do evento

`url` indica a URL em que o evento ocorreu

`key` indica a chave; assume **`null`** se **`clear()`** for chamado

`oldValue` indica o valor anterior; assume **`null`** em uma inserção

`newValue` indica o novo valor; assume **`null`** em uma remoção

`storageArea` é uma referência para o objeto de *storage*
`localStorage` ou **`sessionStorage`**

exemplo

```
// Atenção: o evento NÃO é disparado na aba atual
window.addEventListener( 'storage', function( event ) {
  if ( event.key != 'numero' ) {
    return;
  }
  console.log( 'Número:', event.newValue );
} );

localStorage.setItem( 'numero', Math.random() );
```

observação sobre o evento `storage`

evento permite a comunicação entre frames/abas/janelas

principalmente em navegadores antigos

ou que não suportem à *Broadcast Channel API*

a comunicação pode ocorrer gravando algo no *storage*

disparando evento para a aba/frame atual

somente é possível despachando o evento **manualmente**

porém, ele será recebido **duas vezes** nos outros lugares

exemplo:

```
window.addEventListener( 'storage', function( event ) {
  if ( event.key != 'numero' ) { return; }
  console.log( 'Número:', event.newValue );
} );
localStorage.setItem( 'numero', Math.random() );
// Disparando o evento manualmente
const conteudo = { key: 'numero', newValue: Math.random(), /*...*/};
window.dispatchEvent( new StorageEvent( 'storage' , conteudo ) );
```

referências

MDN. ***Storage Limits***. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria#storage_limits

Web.Dev. **Armazenamento para a web**. Disponível em: https://web.dev/storage-for-the-web/

W3C. **HTML Standard: web storage**. Disponível em: https://html.spec.whatwg.org/multipage/#toc-webstorage

JavaScript.Info. **LocalStorage, sessionStorage**. Disponível em: https://javascript.info/localstorage

fim

2022.10.29 – Versão inicial.