Les fragments

Propriétaire	Ⓜ Marine
Étiquettes	

L'élément `<template>` permet de définir en HTML un fragment de document qui n'est pas rendu lors du chargement de la page, mais peut être cloné et injecté dynamiquement via JavaScript. Il offre une syntaxe claire, une meilleure maintenance du code et des performances optimisées par rapport à la création manuelle d'éléments.

1. Qu'est-ce que `<template>` et à quoi ça sert ?

Le `<template>` est une balise HTML5 destinée à contenir un **DocumentFragment** invisible à l'affichage, prêt à être **cloné** MDN Web Docs.

- **Non rendu** : tout son contenu est ignoré par le moteur de rendu tant qu'on ne l'active pas via JS
- **Clonable** : on y accède via `document.getElementById('id').content`, on clone (`.cloneNode(true)`) et on insère où l'on veut
- **Standard** : supporté par tous les navigateurs modernes depuis 2015

Pourquoi l'utiliser plutôt que de créer les éléments en JS ?

1. **Séparation HTML/JS** : la structure est visible dans le code HTML, plus facile à maintenir MDN Web Docs.
2. **Performance** : le parser HTML construit déjà le fragment, le clonage est plus rapide que la création itinérante d'éléments Stack Overflow.
3. **Lisibilité** : on évite les longues chaînes de création (`createElement`, `appendChild`, etc.).

2. Syntaxe et attributs

```
<template id="card-template">
  <article class="card">
    <h3 class="title"></h3>
    <p class="body"></p>
    <button class="btn">Action</button>
  </article>
</template>
```

3. Accès en JavaScript

```
// 1. Récupérer le template
const tpl = document.getElementById('card-template');

// 2. Accéder au contenu (DocumentFragment)
const frag = tpl.content;

// 3. Cloner en profondeur
const clone = frag.cloneNode(true);

// 4. Personnaliser le clone
clone.querySelector('.title').textContent = 'Mon titre';
clone.querySelector('.body').textContent  = 'Mon contenu...';

// 5. Insérer dans le DOM
document.body.appendChild(clone);
```

4. Exemples d’usage avancés

4.1 Boucles et listes dynamiques

```
<template id="item-template">
  <li class="item"></li>
</template>
<ul id="list"></ul>
```

```
const data = ['Pomme', 'Banane', 'Cerise'];
const tpl  = document.getElementById('item-template');
const list = document.getElementById('list');

data.forEach(text => {
  const clone = tpl.content.cloneNode(true);
  clone.querySelector('.item').textContent = text;
  list.appendChild(clone);
});
```

5. Bonnes pratiques et pièges à éviter

- **Un seul** `<template>` par fragment de structure : évitez d'y mettre trop de responsabilités.
- **Nettoyez** `innerHTML` ou `content` avant chaque clonage si vous réutilisez le même fragment multiple fois.
- **Niveaux d'encapsulation** : pour de gros templates, segmentez en sous-templates.
- **Validation HTML** : le contenu du `<template>` est validé comme du HTML standard