

Коллекции Set&Map.
Асинхронность
(микрос/макрос). Ajax.

Коллекции Set, Map, WeakSet и WeakMap

Set

1. // Коллекция для хранения множества значений, причём
- 2.
3. // каждое значение может встречаться лишь один раз
- 4.
5. `new Set([iterable]);`

Основные свойства и методы

<code>add(item)</code>	добавляет в коллекцию <code>item</code> , возвращает <code>set</code>
<code>delete(item)</code>	удаляет <code>item</code> из коллекции, возвращает <code>true</code> , если он там был
<code>has(item)</code>	возвращает <code>true</code> , если <code>item</code> есть в коллекции
<code>clear()</code>	очищает <code>set</code>
<code>keys()/values()</code>	возвращает объект-итератор на все значения множества
<code>size</code>	возвращает количество элементов

Обход множества

```
1. let mySet = new Set();
2. mySet.add(1); mySet.add("some text");
3. for (let item of mySet) console.log(item);
4. //1, "some text"
5. for (let item of mySet.keys()) console.log(item);
6. //1, "some text"
7. for (let item of mySet.values()) console.log(item);
8. //1, "some text"
9. //преобразует set в Array
0. let myArr = [...mySet];//[1, "some text"]
1. //обратное преобразования
2. let mySet2 = new Set([1,2,3,4]);
3. mySet2.forEach(function(value) {
4.   console.log(value);
5. });
6. //1 2 3 4
```

Микрозадачи/Макрозадачи

Микрозадачи/Макрозадачи

- Цикл событий имеет одну или несколько очередей задач (очередь задач - очередь макрозадачей)
- Каждый цикл событий имеет очередь микрозадачей.
- `task queue = macrotask queue != очередь микрозадачей`
- задача может быть перенесена в очередь макрозаданных или очередь микрозадачей
- когда задача переводится в очередь (микро/макрос), мы подразумеваем, что готовящая работа завершена, поэтому задача может быть выполнена сейчас.

- когда стек вызовов пуст, выполните шаги -
- 1) выберите старую задачу (задачу A) в очередях задач
- 2) если задача A равна null (значит, очереди задач пусты),
- 3) установите "текущая выполняемая задача" в "task A"
- 4) запустите "задачу A" (означает выполнение функции обратного вызова)
- 5) установите "текущая выполняемая задача" на null, удалите "task A"

- **выполнить очередь микрозадач**
 - (a). выберите самую старую задачу (задачу x) в очереди микрозадач
 - (b).if задача x равна нулю (означает, что очереди микрозадач пусты), перейдите к шагу (g)
 - (c).set "текущая выполняемая задача" в "task x"
 - (d).run "task x"
 - (e).set "текущая выполняемая задача" для null, удалить "task x"
 - (f). выберите следующую старую задачу в очереди микрозадач, перейдите к шагу (b)
 - (g). завершить очередь микрозадач;
- **перейдите к шагу 1.**

Проще говоря

- запустите старую задачу в очереди макроса, затем удалите ее.
- запустите все доступные задачи в очереди микрозадач, затем удалите их.
- следующий раунд: запустить следующую задачу в очереди макрозадач (шаг перехода 2)

- когда выполняется задача (в очереди макрозаданных), могут регистрироваться новые события. Таким образом могут быть созданы новые задачи. Ниже приведены две новые созданные задачи:
- `promA.then()` обратный вызов - задача
 - `promA` разрешено/отклонено: задача будет перенесена в очередь микрозадач в текущем раунде цикла событий.
 - `expectedA` ожидает: задача будет перенесена в очередь микрозадач в следующем раунде цикла события (может быть следующий раунд)
- `setTimeout` (обратный вызов, `n`) обратный вызов является задачей и будет перенесен в очередь макрозаданных, даже `n` равно 0:

- в очереди микрозадач будет выполняться в текущем раунде, тогда как задача в очереди макрозадач должна ждать следующего цикла событий.
- Мы все знаем обратный вызов "click", "scroll", "ajax", "setTimeout"... являются задачами, однако мы также должны помнить, что js-коды в целом в теге script - задача (макрозадача) тоже.

- Микрозадачи приходят только из кода. Обычно они создаются промисами: выполнение обработчика `.then/catch/finally`
- Микрозадачи также используются «под капотом» `await`
- Также есть специальная функция `queueMicrotask(func)`, которая помещает `func` в очередь микрозадач.

- `setTimeout(() => console.log("timeout"));`
- `Promise.resolve()`
- `.then(() => console.log("promise"));`
- `console.log("code");`

- `setTimeout(function timeout() {`
- `console.log('Таймаут');`
- `}, 0);`
- `let p = new Promise(function(resolve, reject) {`
- `console.log('Создание промиса');`
- `resolve();`
- `});`
- `p.then(function(){`
- `console.log('Обработка промиса');`
- `});`
- `console.log('Конец скрипта');`

- `setTimeout(() => console.log('setTimeout 1'), 0);`
- `Promise.resolve().then(() => console.log('Promise 1'));`
- `Promise.resolve().then(() => setTimeout(() => console.log('setTimeo
ut 2'), 0));`
- `Promise.resolve().then(() => console.log('Promise 2'));`
- `setTimeout(() => console.log('setTimeout 3'), 0);`
- `console.log('Log 1');`

Ајах.

- АЈАХ (аббревиатура от «**A**synchronous **J**avascript **A**nd **X**ml») – технология обращения к серверу без перезагрузки страницы.
- За счёт этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.
- Несмотря на то, что в названии технологии присутствует буква X (от слова XML), использовать XML вовсе не обязательно. Под АЈАХ подразумевают любое общение с сервером без перезагрузки страницы, организованное при помощи JavaScript.

Ајах. Какие инструменты используются.

- Fetch (начиная с ES6)
- XMLHttpRequest