Поняття про асинхронне програмування

Веб-програмування на стороні сервера 6.1.2024

Синхронне vs Асинхронне виконання

- **Синхронне** виконання передбачає послідовне виконання операцій одна за одною.
  - Допоки попередня операція не буде завершена, наступна не буде розпочата
  - Ми говоримо, що виконання **блокується** до завершення операції

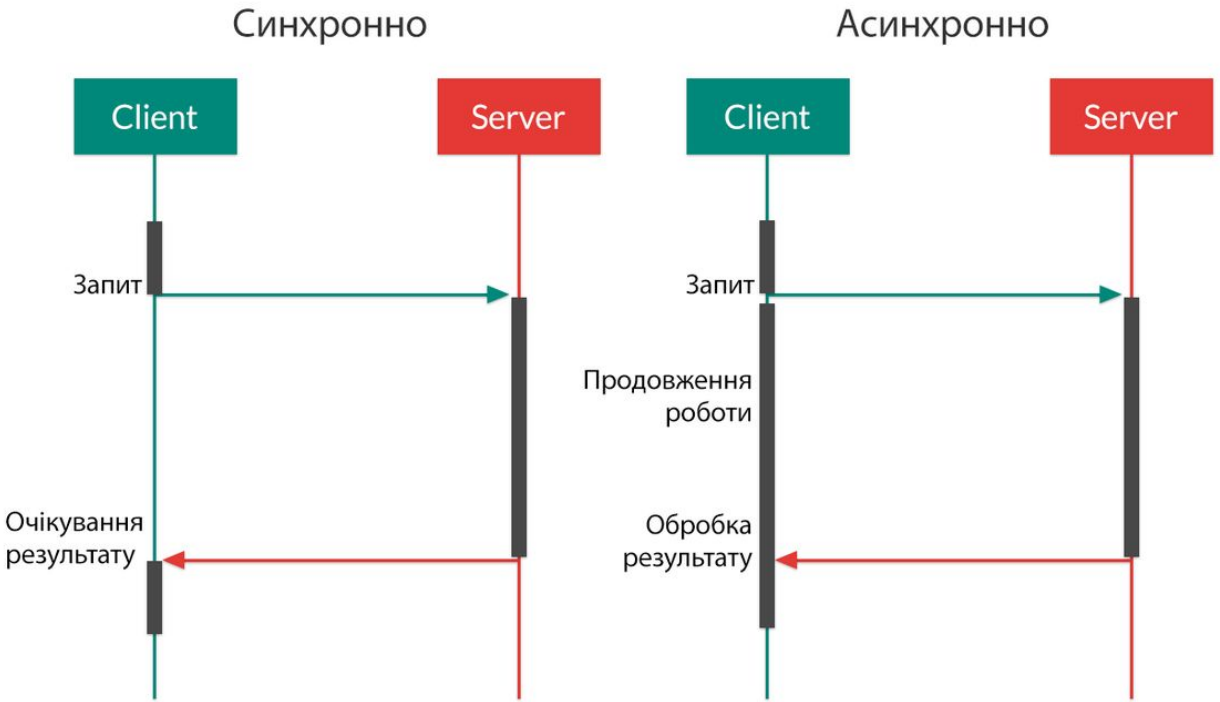
- **Асинхронне** виконання операції передбачає **виконання інших завдань у той час поки операція ще виконується**
  - Операція **не блокує** продовження виконання програми
  - **Ми не можемо передбачити достеменно послідовність виконання, а також коли операція буде завершена**

Коли використовують асинхронне виконання?

- Потреба в асинхронному програмуванні виникає, коли одна частина програми має виконуватися **до**, а інша - **після** довготривалої операції
  - Операції вводу-виводу
  - Мережеві запити
  - Довготривалі обчислення

- **Як ефективно використовувати ресурси, допоки очікується завершення виконання такої операції** - і є суттю асинхронного програмування
  - У той час як ми очікуємо завершення операції - ми можемо виконувати іншу корисну роботу

Синхронне vs Асинхронне виконання: Приклад з запитом на сервер



Асинхронне виконання

- Для асинхронного виконання ми ділимо код на 2 окремі частини:
  - Перша частина підготовує операцію та запускає її на виконання
  - Друга частина опрацьовує результат виконання операції (колбек або проміси)

- Програма має бути побудована таким чином, щоб продовжувати виконувати інші операції допоки не буде відомо результат довготривалої операції яку виконують асинхронно

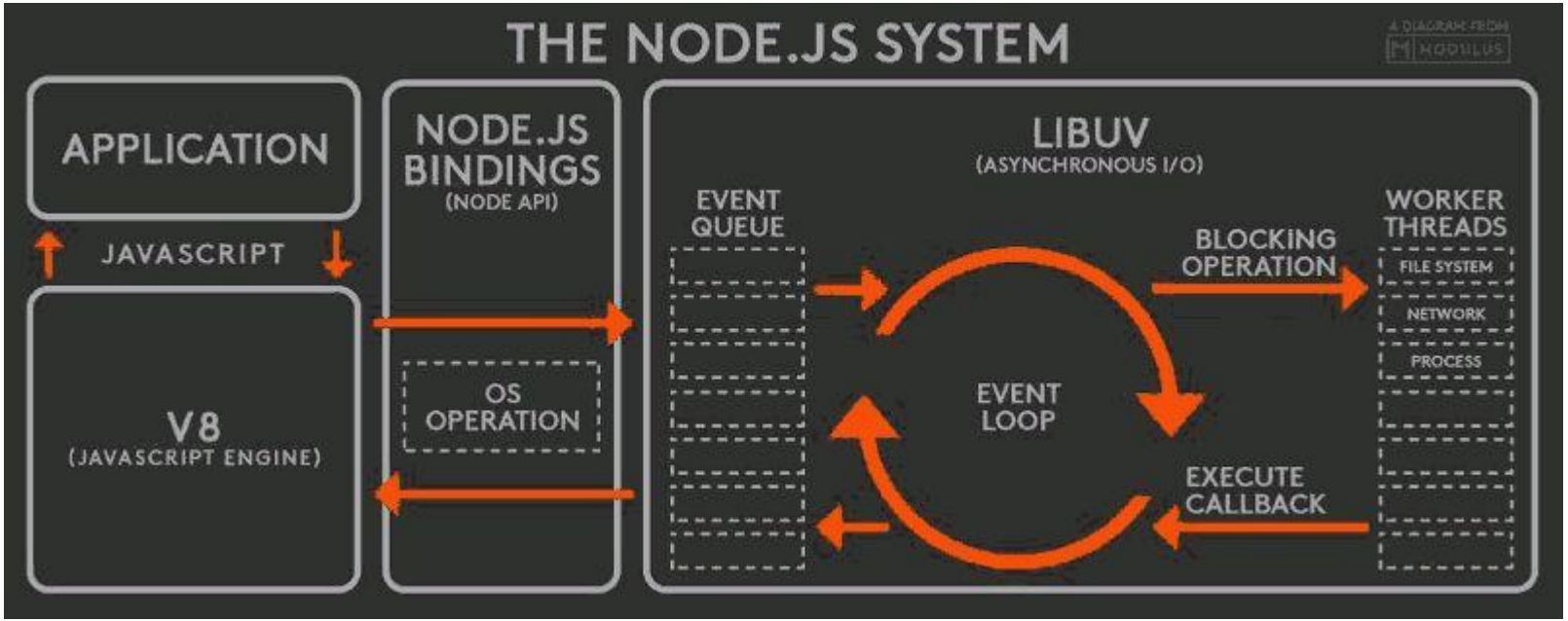
Прийоми асинхронного виконання: Асинхронні колбеки

- Колбек є поширеним прийомом програмування асинхронного виконання завдань. У колбек розміщують код який **буде опрацьовувати результат виконання асинхронної операції**.
- Якщо нам необхідно виконати ланцюжок асинхронних операцій (у якому кожна наступна операція використовує результат попередньої), то можна використати **вкладені колбеки**. Тоді кожна наступна асинхронна операцію запускають на виконання у колбеці, який обробляє результат попередньої.
- Вкладені колбеки можуть сильно поскладнювати розуміння логіки програми і її відлагодження. У крайніх випадках говорять про “пекло колбеків” (callback hell, http://callbackhell.com/)

Прийоми асинхронного виконання: Асинхронні колбеки

- Існують кілька прийомів, щоб зробити програму легшою для розуміння та відлагодження:
  - Передавати 2 окремі колбеки: один - для успішного випадку і другий - для обробки помилок
  - Резервувати окремий параметр колбеку для об’єкта помилки - так як це зазвичай роблять у вбудованих модулях Node.js (“error-first style”)
  - Створювати окремі колбеки для синхронного і окремі для асинхронного виконання - уникати ситуацій коли один і той же колбек буде викликатися часом синхронно, а часом асинхронно
  - Використовувати інші прийоми замість асинхронних колбеків

Спрощена схема внутрішньої будови Node.js



https://dou.ua/forums/topic/31698/

Деякі функції, які дозволяють запланувати виконання операції у Node.js

- setTimeout() / clearTimeout() - запланувати операцію з затримкою
- setImmediate() - запланувати операцію одразу
- setInterval() - запланувати повторювану задачу

https://nodejs.org/uk/docs/guides/timers-in-node

Прийоми асинхронного виконання: Проміси

- Хороша метафора - чек замовлення у МакДональдс. Чек ви отримуєте одразу, але замовлення ви можете отримати за чеком і з'їсти тільки коли воно буде готове. Про виконання замовлення вам повідомляють.

https://hips.hearstapps.com/hmg-prod/images/mcds-1554904333.jpg

Прийоми асинхронного виконання: Проміси

- Проміс репрезентує певне **майбутнє значення/результат**, яке ще може бути невідоме (бо операція, яка поверне це значення, ще не завершена)
  - Результатом може бути також помилка, яка виникла у результаті виконання операції
- Має наступні можливості:
  - Задати функцію виконавця
  - Задати функцію споживача
  - Задати функцію для обробки помилок
  - Задати функцію для фіналізації операції

Проміси

- Проміс репрезентує певне **майбутнє значення/результат**, яке ще може бути невідоме (бо операція, яка поверне це значення, ще не завершена)
  - Результатом може бути також помилка, яка виникла у результаті виконання операції
- Має наступні можливості:
  - Задати функцію виконавця
  - Задати функцію споживача
  - Задати функцію для обробки помилок
  - Задати функцію для фіналізації операції

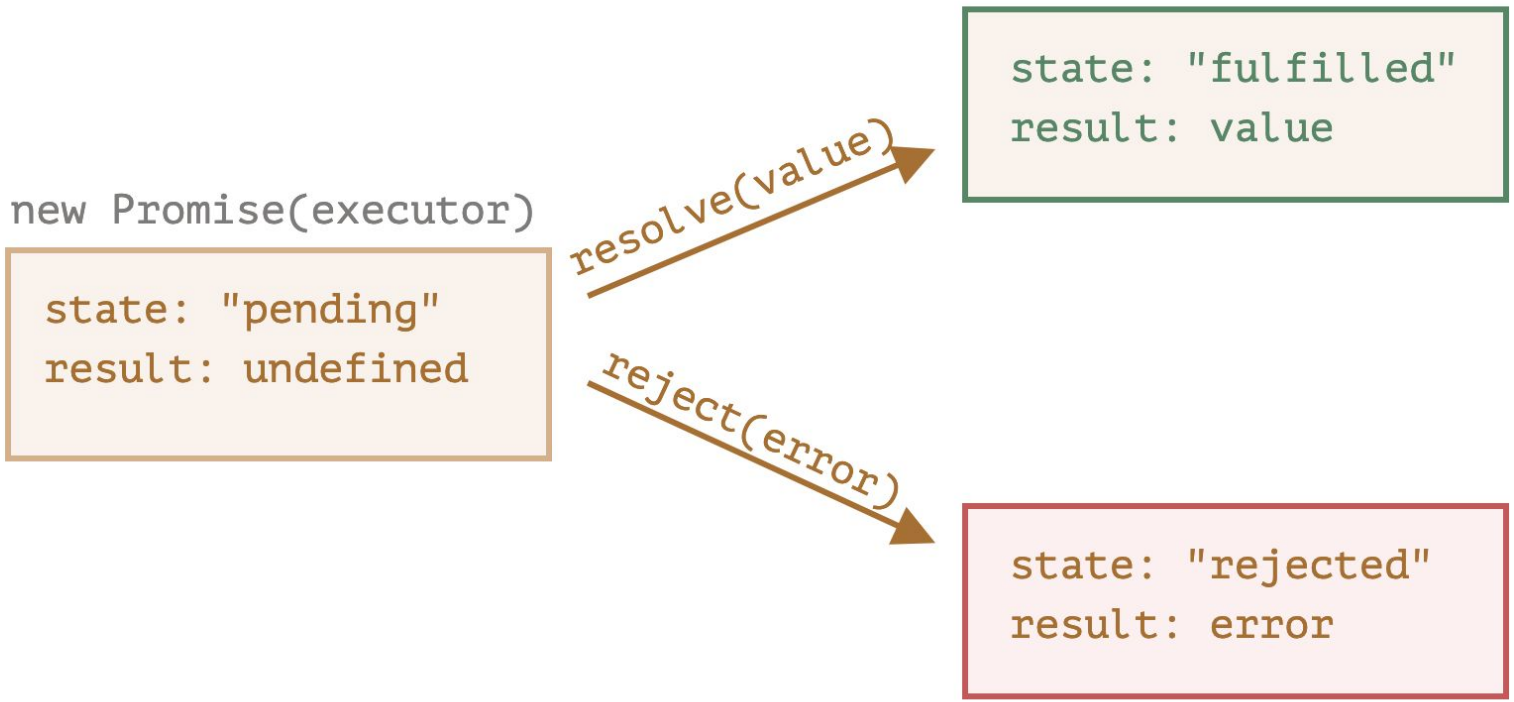
Проміси

```
let promise = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1 second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
});
```

```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

Проміси


new Promise(executor)
state: "pending"
result: undefined
resolve(value)
reject(error)
state: "fulfilled"
result: value
state: "rejected"
result: error

Проміси

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);
```

```
new Promise((resolve, reject) => {
  throw new Error("error");
})
  .finally(() => alert("Promise ready")) // triggers first
  .catch(err => alert(err));  // <-- .catch shows the error
```