Java Script

NEXT

План

Асинхронний Js

Синхронність, асинхронність, багатопотоковість

setTimeout и setInterval

Дата

Promise

GENIUS space

Асинхронний JavaScript

При розробці часто доводиться мати справу з асинхронною поведінкою, яка може бентежити новачків, які мають досвід роботи з синхронним кодом.
Синхронний код виконується послідовно, кожна інструкція чекає перед виконанням, поки виконається попередня.
Асинхронний код виконується, не чекаючи виконання попередніх інструкцій.

Більшість функціональності, яку ми розглядали в попередніх навчальних модулях, є синхронною - ви запускаєте якийсь код, а результат повертається, як тільки браузер може його повернути.

З причин, згаданих раніше (наприклад, що стосуються блокування), безліч Web API особливостей тепер використовують асинхронний код, особливо ті, що мають доступ до зовнішніх пристроїв або отримують від них деякі ресурси, такі як отримання файлу з мережі, запит до бази даних та отримання даних із бази, доступ до потокового відео на веб-камері, перегляд дисплея на гарнітурі віртуальної реальності.

GENIUS space

Чому важко працювати за допомогою синхронного коду? Погляньмо на невеликий приклад. Коли ви отримуєте зображення з сервера, ви не можете миттєво повернути результат. Це означає, що наступний (псевдо) код не спрацює

```
let response = fetch('myImage.png');l
et blob = response.blob();
// display your image blob in the UI somehow
```

Це відбувається тому, що ви не знаєте скільки часу займе завантаження картинки, отже, коли ви почнете виконувати другий рядок коду, згенерується помилка (можливо, періодично, можливо, щоразу), тому що response ще не доступний. Натомість, ваш код повинен дочекатися повернення response до того, як спробує виконати подальші інструкції.

Є два типи стилю асинхронного коду, з якими ви зіткнетеся в коді JavaScript, старий метод - колбеки (callbacks) і новий - проміси (promises).

GENIUS space

Зверніть увагу, що асинхронний - це не те ж саме, що й одночасний або багатопотоковий. JavaScript може мати асинхронний код, але він, як правило, однопотоковий. Це як в ресторані з одним працівником, який робить все: подача і приготування їжі. Але якщо цей працівник працює досить швидко і може перемикатися між завданнями досить ефективно, то буде здаватися, що у ресторані є кілька робітників.

В асинхронних багатопотокових процесах ви призначаєте завдання працівникам. В асинхронних однопотокових процесах у вас є графік завдань, де деякі завдання залежать від інших. У міру виконання кожного завдання викликається код, який планує наступне завдання, яке може бути запущено, з урахуванням результатів щойно виконаного завдання. Але вам потрібний лише один працівник для виконання всіх завдань, а не один працівник на завдання.

GENIUS space

setTimeout и setInterval

Ми можемо вирішити виконати функцію не зараз, а через певний час пізніше. Це називається “планування виклику”.

Для цього існує два методи:

setTimeout дозволяє нам запускати функцію один раз через певний інтервал часу.
setInterval дозволяє нам запускати функцію багаторазово, починаючи через певний інтервал часу, а потім постійно повторюючи у цьому інтервалі.
Ці методи не є частиною специфікації JavaScript. Але більшість середовищ виконання JS-коду мають внутрішній планувальник і надають ці методи. Зокрема, вони підтримуються у всіх браузерах та Node.js.

GENIUS space

setTimeout
Синтаксис:

let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)
Параметри:

func|code
Функція або рядок коду для виконання. Зазвичай це функція. З історичних причин можна передати рядок коду, але це не рекомендується.
delay
Затримка перед запуском, у мілісекундах (1000 мс = 1 секунда), типове значення – 0.
arg1, arg2...
Аргументи, які передаються у функцію (не підтримується в IE9-)

GENIUS space

Скасування за допомогою clearTimeout

Виклик setTimeout повертає “ідентифікатор таймера” timerId, який ми можемо використовувати для скасування виконання.

Синтаксис для скасування:

```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

У наведеному нижче коді ми плануємо функцію, а потім скасовуємо її (просто передумали). В результаті нічого не відбувається:

```
let timerId = setTimeout(() => alert("ніколи не відбувається"), 1000);
alert(timerId); // ідентифікатор таймера

clearTimeout(timerId);
alert(timerId); // той самий ідентифікатор (не приймає значення null після скасування)
```

Як ми бачимо з результату alert, в браузері ідентифікатор таймера – це число. В інших середовищах це може бути щось інше. Наприклад, Node.js повертає об’єкт таймера з додатковими методами.

GENIUS space

setInterval
Метод setInterval має той самий синтаксис, що і setTimeout:

let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
Усі аргументи мають такі ж самі значення. Але на відміну від setTimeout, цей метод запускає функцію не один раз, але регулярно через заданий проміжок часу.

Щоб припинити подальші виклики, нам слід викликати clearInterval (timerId).

Наступний приклад буде відображати повідомлення кожні 2 секунди. Через 5 секунд вивід припиняється:

```
// повторити з інтервалом 2 секунди
let timerId = setInterval(() => alert('тік'), 2000);

// зупинити через 5 секунд
setTimeout(() => { clearInterval(timerId); ale
```

GENIUS space

Date - об'єкт для роботи з датою та часом.

Синтаксис:
new Date();
new Date(value);
new Date(year, month, date[, hours[, minutes[, seconds[,ms]]]]);
Параметри:
value - значення в якому задається дата і час. Значення буває: числове і рядкове.
Якщо значення числове то вказується кількість мілісекунд від 01.01.1970 00:00:00.
Якщо значення рядкове то вказується рядкове предсавлення дати і часу у форматі який розпізнає метод Date.parse().

year - Обов'язковий параметр. Число яке вказує рік.
У різних браузерах рік вказується по різному: двох значними числами 00 до 99 і чотирьох значними числами.
Рекомендовано вказувати чотирьох значне число, наприклад: 2016. Для сумісності з браузерами.

GENIUS space

month - Обов'язковий параметр.

Вказується місяць від 0 до 11, де:
0 - січень
1 - лютий
2 - березень
3 - квітень
4 - травень
5 - червень
6 - липень
7 - серпень
8 - вересень
9 - жовтень
10 - листопад
11 - грудень

GENIUS space

date - Обов'язковий параметр. День від 1 до 31.

hours - година від 0 до 23.
minutes - хвилина від 0 до 59.
seconds - секунди від 0 до 59.
ms - мілісекунди від 0 до 999.

Об'єкт Date служить для отримання дати і часу і для роботи з датою і часом.
При створенні об'єкту Date використання оператора new є обов'язкове. Інакше об'єкт не буде створено а повернеться рядок з поточною датою.

Якщо при створені об'єкту Date не вказано жодного параметру то об'єкту присвоюється поточна системна дата і час.
У **JavaScript** дата і час вимірюється у мілісекундах які пройшли з 01.01.1970 по UTC (Всесвітній координований час).
Місцевий час у JavaScript це час на локальному комп'ютері, на якому виконується JavaScript.

GENIUS space

Властивості об'єкта Date:

Date.length- кількість параметрів.

Методи об'єкта Date:

Date.now() - число мілісекунд з 01.01.1970.

Date.parse() - розбирає рядкове представленя дати і повертає у мілісекундах.

Date.UTC() - повертає мілісекунди у вказаній даті.

Date.getDate() - повертає день.

Date.getDay() - повертає номер дня у тижні.

Date.getFullYear() - повертає рік.

Date.getHours() - повертає годину

Date.getMonth() - повертає місяць.

Date.getMinutes() - повертає хвилини.

- Date.getSeconds() - повертає секунди.
- Date.getMilliseconds() - повертає мілісекунди.
- Date.getTime() - повертає кількість мілісекунд у даті.
- Date.getTimezoneOffset() - зміщення часового поясу у хвилинах.
- Date.getUTCDate() - повертає день у форматі UTC.

GENIUS space

Date.getUTCDay() - повертає номер дня у тижні у форматі UTC.
Date.getUTCFullYear() - повертає рік у форматі UTC.
Date.getUTCHours() - повертає годину у форматі UTC.
Date.getVarDate() - повертає значення VT_DATE.
Date.getYear() - повертає значення рік-1990.
Date.setDate - встановлює день місяця.
Date.setFullYear - встановлює рік.
Date.setHours - встановлює годину.
Date.setMinutes() - встановлює хвилину.
Date.setMonth() - встановлює місяць.
Date.setTime() - встановлює дату по вказаним мілісекундам.
Date.setUTCDate() - встановлює день місяця по UTC.
Date.setUTCFullYear() - встановлює рік по UTC.
Data.setUTCHours - встановлює годину по UTC.

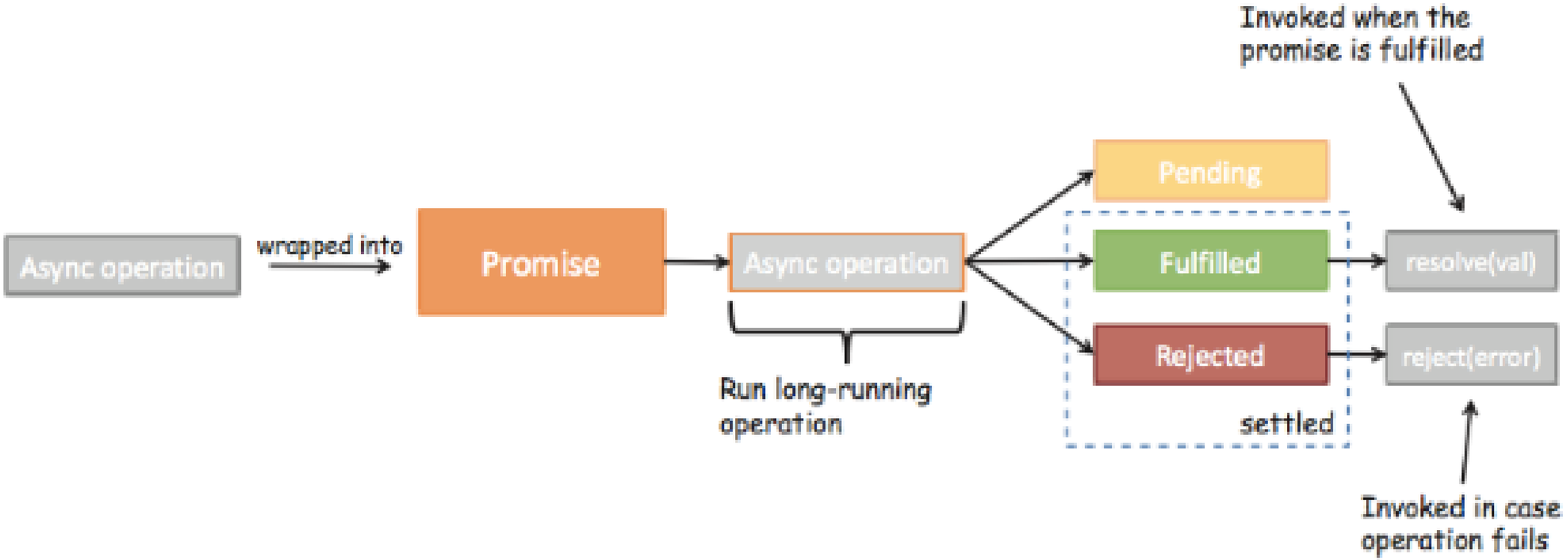
GENIUS space

Data.setUTCMilliseconds - встановлює мілісекунди по UTC.
Date.setUTCMinutes() - встановлює хвилини по UTC
Date.setYear() - встановлює рік.
Date.toDateString() - повертає рядок з датою.
Date.toGMTString() - повертає рядок з датою і часом по GMT.
Date.toJSON - повертає дату і час у рядку відформатовану для JSON.
Date.toLocaleDateString() - повертає рядок з лише дату по локалі.
Date.toLocaleString() - повертає рядок з датою і часом по локалі.
Date.toLocaleTimeString() - повертає рядок з лише годиною по локалі.
Date.toSource() - повертає рядок з вихідним кодом об'єкту.
Date.toString() - повертає рядок з датою і часом.
Date.toTimeString() - повертає рядок який містить тільки годину.
Date.toUTCString() - повертає рядок який містить дату і час по UTC.
Date.valueOf() - повертає примітивне значення об'єкту Data.

GENIUS space

Promise (обіцянка, проміс) - об'єкт, що представляє поточний стан асинхронної операції. Зручний спосіб організації асинхронного коду.
У промісу є 2 стани:
Pending - очікування, вихідний стан під час створення промісу.
Settled – виконаний, яке у свою чергу ділиться на дві категорії: fullfilled – виконано успішно та rejected – виконано з помилкою.
Спочатку проміс перебуває у стані очікування (pending), після чого може виконатися успішно (fulfilled) чи з помилкою (rejected). Коли проміс перетворюється на стан виконаний (settled), з результатом чи помилкою – це назавжди. Грубо кажучи, проміс - це заглушка для даних, значення яких ми не знаємо в момент його створення.
Спосіб використання:
Код, якому треба зробити щось асинхронно, створює обіцянку та повертає її.
Зовнішній код, отримавши обіцянку, навішує на нього оброблювачі.
По завершенні процесу асинхронний код переводить обіцянку в стан fulfilled або rejected. При цьому автоматично викликаються обробники у зовнішньому коді.

GENIUS space


Async operation
wrapped into
Promise
Async operation
Run long-running operation
Pending
Fulfilled
Rejected
settled
Invoked when the promise is fulfilled
resolve(val)
reject(error)
Invoked in case operation fails

Властивості
Promise.length
Значення якості завжди дорівнює 1 (кількість аргументів конструктора).

Promise.prototype (en-US)
Прототип для конструктора Promise.

Методи

Promise.all(iterable)
Чекає на виконання всіх промісів або відхилення будь-якого з них.

Повертає проміс, який здійсниться після виконання всіх промісів у iterable. У випадку, якщо будь-який з промісів буде відхилено, Promise.all також буде відхилено.

GENIUS space

Promise.allSettled(iterable)
Чекає на завершення всіх отриманих промісів (як виконання так і відхилення).

Повертає проміс, який виконується, коли всі отримані проміси завершені (виконані або відхилені), що містить масив результатів виконання отриманих промісів.

Promise.race(iterable)
Чекає на виконання або відхилення будь-якого з отриманих промісів.

Повертає проміс, який буде виконаний або відхилений з результатом виконання першого виконаного або відхиленого промісу з iterable.

Promise.reject(reason)
Повертає проміс, відхилений через reason.

Promise.resolve(value)
Повертає проміс, виконаний результатом value.

GENIUS space

Синтаксис створення проміса:

lОбіцянка створюється як екземпляр класу Promise з однією функцією як аргумент. Виклик конструктора негайно виконає функцію fn, передану як аргумент. Мета цієї функції полягає в інформуванні екземпляра (промісу), коли подія, з якою він пов'язаний, буде завершено.

```
const promise = new Promise((resolve, reject) => {
  /*
   * Ця функція буде викликана автоматично. У ній можна виконувати
   * будь-які асинхронні операції. Коли вони завершаться – потрібно
   * викликати одне з: resolve(результат) при успішному виконанні,
   * або reject(помилка) при помилці.
   */
});
```

GENIUS space

Використання

Після того, як проміс створено, з ним можна працювати використовуючи методи then та catch, які доступні через його прототип. Код пишеться так, ніби ми розмірковуємо про те, що може статися, якщо проміс виконається чи ні, не думаючи про тимчасові рамках

then
Дозволяє виконати код, в якому можна отримати доступ і обробити результат промісу.
promise.then(onResolve, onReject)

У метод передаються дві функції які будуть викликані коли проміс перейде в стан виконаний (settled).
onResolve(arg) - буде викликана при успішному виконанні промісу, і отримає результат промісу як аргумент (те, що передаємо виклик resolve).
onReject(arg) - буде викликана при виконанні промісу з помилкою, і отримає помилку як аргумент (те, що передаємо виклик reject).

GENIUS space

catch

Трохи далі ми дізнаємося про ланцюжки промісів, а поки що навчимося обробляти помилки не в колбеку на reject методу then, а в спеціальному методі catch. Обробляти помилки дуже зручно, використовуючи метод catch тільки один раз, в кінці ланцюжка.

promise.catch(onReject)

Хендлер для обробки стану reject виконається тільки якщо проміс виконається з помилкою (rejected). onReject(arg) буде викликана при виконанні промісу з помилкою і отримає помилку як аргумент (те, що передаємо у виклик reject).

GENIUS space

finally

Цей метод може бути корисним, якщо ви хочете виконати деяку обробку або очищення після того, як обіцянка буде виконана, незалежно від результату. Дозволяє виконати вказану callback-функцію після того, як обіцянку буде дозволено (виконано або відхилено). Дозволяє уникнути дублювання коду в обробниках then() та catch(). Повертає обіцянку.

```
promise.finally(() => {
  // settled (fulfilled або rejected)
});
```

Функція зворотного дзвінка не отримає жодних аргументів, оскільки не можна точно визначити, чи виконано обіцянку або відхилено. Тут буде виконуватися код, який залежить тільки від часу його виконання, значення промісу не важливо.

GENIUS space

Додаткові матеріали

- с https://codeguida.com/post/445
- https://uk.javascript.info/settimeout-setinterval
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
- https://flaviocopes.com/javascript-dates/
- https://flaviocopes.com/momentjs/
- https://momentjs.com/
- https://uk.javascript.info/promise-basics

GENIUS space

Домашнє завдання

-----1----

Напишіть функцію printNumbers(from, to) яка виводить число кожну секунду, починаючи від from і закінчуючи to.
Зробіть два варіанти рішення.
Використовуючи setInterval.
Використовуючи вкладений setTimeout

-----2----

Вбудована функція setTimeout використовує колбек-функції. Створіть альтернативу яка базується на промісах.
Функція delay(ms) повинна повертати проміс, який перейде в стан resolved через ms мілісекунд, так щоб ми могли додати до нього .then:

```
function delay(ms) {
// ваш код
}delay(3000).then(() => alert('виконалось через 3 секунди'));
```

GENIUS space