[Faraz Kelhini](https://blog.logrocket.com/author/farazkelhini/)JavaScript-разработчик

**6 новейших функций JavaScript, которые ты можешь использовать уже сегодня**

Сейчас самое лучшее время для Java-программистов. Веб-технологии быстро развиваются и разработчики браузеров без колебаний внедряют новые передовые свойства. Взрывное развитие требует от программиста постоянного улучшения своих навыков, чтобы оставаться конкурентноспособным в своей профессии.

В это статье мы рассмотрим шесть ES2020 и ES2021 функций, которые были недавно внедрены современными браузерами, и увидим, как они помогают JavaScript-разработчикам эффективный код, который менее подвержен ошибкам.

BigInt

Когда мы работаем с большими целыми числами в JavaScript, нам часто приходится использовать сторонние библиотеки, потому что тип Number не может коректно представлять значения целых чисел больше, чем $2^{53}$.

Рассмотрим на следующем примере:

console.log(9999999999999999); // => 10000000000000000

В этом коде число 9999999999999999 округляется до 10000000000000000 , потому что оно больше за максимальное целое число, поддерживаемое типом Number. Если вы будете невнимательны, такое округление может поставить под угрозу работу вашей программы.

Вот следующий пример:

// notice the last digit

9800000000000007 === 9800000000000008; // => true

К счастью, ECMAScript недавно представил тип данных BigInt, который обеспечивает простой способ представления целых чисел, превышающих диапазон, поддерживаемый типом Number .

Мы можем создать BigInt , добавляя n к целому числу.

Сравним:

console.log(9800000000000007n); // => 9800000000000007n

console.log(9800000000000007); // => 9800000000000008

Также можно использовать конструктор:

BigInt('9800000000000007'); // => 9800000000000007n

Теперь вы можете выполнять арифметические операции с большими целыми числами в стандартном JavaScript без дополнительных усилий:

9999999999999999 \* 3; // => 30000000000000000

// с BigInt, округление целых чисел не будет проблемой

9999999999999999n \* 3n; // => 29999999999999997n

Важно понимать, что Number и BigInt - это два разных типа данных, и вы не можете сравнивать их с оператором строгого равенства:

5n === 5; // => false

typeof 5n; // => bigint

typeof 5; // => number

Однако вы все равно можете использовать оператор равенства, потому что он неявно преобразует операнды в один и тот же тип перед сравнением:

5n == 5; // => true

Вы можете выполнять арифметические операции с BigInt точно так же, как с Number:

50n + 30n; // => 80n

50n - 30n; // => 20n

50n \* 20n; // => 1000n

50n / 5n; // => 10n

56n % 10n; // => 6n

50n \*\* 4n; // => 6250000n

Операторы инкремента, декремента и унарного отрицания также работают должным образом. Но унарный оператор плюс (+) является исключением, и его применение к BigInt вызовет ошибку TypeError:

let x = 50n;

++x; // => 51n

--x; // => 50n

-50n; // => -50n

+50n; // => TypeError: Невозможно преобразовать значение BigInt в число

**Оператор объединения с нулевым значением.**

ES2020 добавляет еще один оператор сокращения в язык JavaScript: оператор объединения с нулевым значением (??).Этот оператор отличается от существующих сокращающих операторов тем, что он проверяет, является ли его левый операнд нулевым (null или undefined), а не ложным.

Другими словами, ?? возвращает свой правый операнд, только если его левый операнд равен null или undefined:

null ?? 2; // => 2

undefined ?? 2; // => 2

0 ?? 2; // => 0

false ?? true; // => false

С другой стороны, логический оператор OR (||) возвращает свой правый операнд, если левый операнд равен 0, -0, 0n, false, "" (пустая строка), null, undefined или NaN.

Сравним:

null || 2; // => 2

undefined || 2; // => 2

0 || 2; // => 2

false || true; // => true

?? особенно удобен при установке значения по умолчанию для свойства или переменной. Например:

function Config(darkMode) {

 this.darkMode = darkMode ?? true;

 // …

}

new Config(null); // => {darkMode: true}

new Config(); // => {darkMode: true}

new Config(false); // => {darkMode: false}

Конструктор Config предоставляет значение по умолчанию для свойства darkMode в случае, если заданное значение равно нулю или значение не задано.

?? также полезно при работе с DOM API:

// querySelector() returns null when the element doesn’t exist in the document

const elem = document.querySelector('elem') ?? document.createElement('elem');

Имейте в виду, что при использовании ?? с другими упрощенными операторами в выражении вы должны обозначать порядок вычисления круглыми скобками, иначе код выдаст ошибку.

Круглые скобки также помогают читабельности кода:

false || (true ?? true); // no error

false || true ?? true; // => SyntaxError

**Promise.any()**

ES2015 представил объект обещания с двумя методами: Promise.all() и Promise.race(). ES2021 дополнительно расширяет асинхронные возможности JavaScript, добавляя Promise.any().Этот новый метод возвращает обещание, которое выполняется, когда выполняется одно из обещаний в данной итерации, или отклоняет, если все обещания отклоняются.

Вот как это работает:

const promise = Promise.any([

 Promise.reject('Error'),

 fetch('https://www.google.com/images/branding/googlelogo/2x/googlelogo\_color\_92x30dp.png', {mode: 'no-cors'}).then(() => 'google.com'),

 fetch('https://en.wikipedia.org/static/images/project-logos/enwiki.png', {mode: 'no-cors'}).then(() => 'wikipedia.org'),

 fetch('https://s.w.org/images/home/swag\_col-1.jpg?1', {mode: 'no-cors'}).then(() => 'w.org')

]);

promise.then((fastest) => {

 // the first promise that fulfills

 console.log(fastest);

}).catch((error) => {

 console.log(error);

});

Этот код выполняет три запроса на выборку. Как только одно из обещаний выполнено, оно возвращает обещание, которое выполняется со значением из этого обещания. Promise.any() отличается от Promise.race() тем, как он обрабатывает отклонение. Обещание, возвращаемое Promise.any(), отклоняется только в том случае, если все обещания в повторяемом отклоняются:

const promise = Promise.any([

 Promise.reject('Exception1'),

 Promise.reject('Exception2'),

 Promise.reject('Exception3')

]);

promise.then((response) => {

 // ...

}).catch((e) => {

 console.log(e.errors);

});

// logs:

// => ["Exception1", "Exception2", "Exception3"]

Обратите внимание, как значение отклонения всех обещаний передается в метод catch() в виде массива. В качестве альтернативы вы можете использовать async и await для обработки результата:

(async () => {

 try {

 result = await Promise.any([

 Promise.reject('Exception1'),

 Promise.reject('Exception2'),

 Promise.resolve('Success!')

 ]);

 console.log(result);

 } catch(e) {

 console.log(e);

 }

})();

// logs:

// => Success!

## Promise.allSettled()

Еще один полезный метод, который недавно был добавлен к объекту обещания, - это Promise.allSettled(). Этот метод, дополняющий существующий метод Promise.all(), предназначен для возврата результата всех обещаний - как отклоненных, так и выполненных.

Вот пример:

const p1 = Promise.resolve('Success');

const p2 = Promise.reject('Exception');

const p3 = Promise.resolve(123);

Promise.allSettled([p1, p2, p3]).then((response) => {

 response.forEach(result => console.log(result.value || result.reason))

});

// logs:

// => Success

// => Error!

// => 123

Обратите внимание, как результат всех обещаний передается в then() в виде массива. Внутри then() метод forEach() перебирает элементы массива. Если левый операнд || оператора не является неопределенным, он будет записан в консоль. В противном случае обещание было отклонено, и правильный операнд будет зарегистрирован.

Для сравнения, Promise.all() немедленно отклоняет, как только одно из обещаний отклоняется.

## Оператор опциональной последовательности

Оператор опциональной последовательности (?.) Позволяет получить доступ к вложенному свойству без проверки каждого свойства в последовательности.

Рассмотрим следующий пример:

const obj = {};

const nickname = obj?.user?.profile?.nickname;

console.log(nickname); // => undefined

Этот код пытается присвоить значение вложенного свойства константе. Но такого свойства в obj. не существует. Кроме того, user и profile не существует. Но благодаря оператору опциональной последовательности код возвращает undefined вместо выдачи ошибки.

Используя оператор обычной последовательности, вы получите ошибку:

const obj = {};

const nickname = obj.user.profile.nickname;

console.log(nickname); // => TypeError

Оператор опциональной последовательности также можно использовать при вызове метода объекта:

const obj = {};

const value = obj.myMethod?.();

console.log(value); // => undefined

Здесь myMethod не существует в obj; однако, поскольку он вызывается с использованием оператора опциональной последовательности, возвращаемое значение не определено. Опять же, с оператором обычной последовательности вы получите ошибку.

Но что, если вы хотите получить доступ к свойству динамически? ?.[] позволяет ссылаться на переменную в скобках.

Вот как это работает:

const obj = {

 user: {

 id: 123

 }

};

const prop = 'nickname';

const nickname = obj?.user?.profile?.[prop];

console.log(nickname); // => undefined

## globalThis

Хотя JavaScript был создан с целью выполнения сложных функций в веб-браузерах, теперь он может работать в совершенно разных средах, таких как серверы, смартфоны и даже роботы. Поскольку каждая среда имеет свою собственную объектную модель, вам потребуется использовать другой синтаксис для доступа к глобальному объекту.

В среде браузера вы можете использовать window, frames или self. В Web Workers вы можете использовать self. А в Node вы можете использовать global. Это несоответствие усложняет веб-разработчикам написание совместимых программ на JavaScript.

globalThis обеспечивает единое универсальное свойство во всех средах для доступа к глобальному объекту:

// browser environment

console.log(globalThis); // => Window {...}

// web worker environment

console.log(globalThis); // => DedicatedWorkerGlobalScope {...}

// node environment

console.log(globalThis); // => Object [global] {...}

Раньше разработчикам приходилось писать дополнительные проверки, чтобы убедиться, что они ссылаются на правильное свойство. С globalThis это больше не требуется, и код будет работать как в оконном, так и в не оконном контекстах. Имейте в виду, что вам все равно может потребоваться использовать полифилл для обратной совместимости в старых браузерах.

## Вывод

JavaScript быстро развивается, и время от времени в язык добавляются новые интересные функции. В этой статье мы рассмотрели шесть новых функций JavaScript, включая BigInt, оператор объединения с нулевым значением, Promise.any(), Promise.allSettled(), оператор опциональной последовательности и globalThis.

BigInt позволяет представлять большие целочисленные значения. Оператор объединения с нулевым значением добавляет в JavaScript новый оператор минимального вычисления. Promise.any() и Promise.allSettled() позволяют дополнительно контролировать асинхронные операции. Оператор опциональной последовательности упрощает доступ к вложенным свойствам. И globalThis предоставляет единое универсальное свойство для всех сред для доступа к глобальному объекту.

Чтобы быть в курсе последних функций, просмотрите список готовых предложений. Если у вас есть какие-либо вопросы, не стесняйтесь задавать их в комментариях, мой Твиттер.