7.4. async/await

~ 6 минут

В прошлом разделе мы познакомились с промисами и узнали, как они упрощают написание асинхронного кода. Строить цепочки, вместо «лестниц» колбэк-функций намного удобней и поддерживать такой код проще. В этом разделе разберёмся с операторами `async/await`. С их помощью писать асинхронный код станет ещё проще.

А как же промисы

Прочитав введение к этому разделу, вы справедливо могли подумать, что промисы нам больше не потребуются. Зачем тогда мы уделили им столько времени? Не переживайте. Полученные знания не пропадут даром. В основе `async/await` лежат промисы. `async/await` — это синтаксический сахар для промисов, который делает ваш асинхронный код ещё более похожим на синхронный.

Чтобы начать применять `async/await`, важно разобраться с промисами, поэтому, если вы пропустили прошлый раздел, самое время вернуться.

async

Начнём знакомиться с введения нового ключевого слова — `async`. Оно позволяет определить асинхронную функцию. Результатом выполнения такой функции будет новый промис. `async` вы можете использовать как при объявлении функций декларативно, так и для функций-выражений. Рассмотрим сразу несколько примеров:

```
async function foo() {
  // Применение async для функции объявленной декларативно
}

const boo = async () => {
  // Применение async для функции объявленной в виде выражения
}
```

```
console.log(foo()); // Promise
console.log(boo()); // Promise
```

Результатом выполнения двух функций станет новый промис. Мы всего лишь добавили одно ключевое слово и получили промис. Если применять промисы, то нам пришлось бы воспользоваться конструктором, либо выполнить `resolve`. Давайте попробуем немного улучшить пример и вернуть из `async` функции какое-нибудь значение. По традиции вернём фразу `Hello, world`, а затем сделаем вывод в консоль:

```
const foo = async () => 'Hello, world';

foo()
 .then((text) => console.log(text)); // Hello, world
```

В примере мы воспользовались сокращённым синтаксисом применения стрелочных функций. Раз в теле функции сразу возвращаем результат, то необязательно применять фигурные скобки.

Таким образом, функция `foo` вовсе превращается в однострочную. Вы уже знаете, результатом выполнения `async` функции будет новый промис. Внутри функции возвращается строка `Hello, world`, как если бы воспользовались методом `resolve`. Дальше применяем уже знакомый метод `then`, передав ему колбэк-функцию.

Рассмотренный пример — это ни что иное как применение статического метода `resolve` у объекта `Promise`. Давайте посмотрим на вариант синтаксиса с промисами, чтобы проще было провести аналогию. То же самое:

```
const foo = () => Promise.resolve('Hello, world');

foo()
 .then((text) => console.log(text));
```

await

Ключевое слово `async` избавляет от необходимости создавать и возвращать промисы. Эта операция выполняется автоматически. Теперь посмотрим на второе ключевое слово — `await`. Оператор позволяет дождаться окончания выполнения промиса.

Важно сразу запомнить: ключевое слово `await` допускается использовать только внутри функции, объявленной с использованием ключевого слово `async`. Рассмотрим на примере получения списка публикаций:

```
const URL = 'https://jsonplaceholder.typicode.com/posts';
```

```
const URL = 'https://jsonplaceholder.typicode.com/posts';

const getPosts = async () => {
  const response = await fetch(URL);
  const posts = await response.json();
  return posts;
};

(async () => {
    const posts = await getPosts()
    console.log('Список публикаций: ');
    console.log(posts);
})();
```

Обратите внимание, функция `getPosts` объявлена с использованием ключевого слова `async`, то есть результатом функции станет промис. В первой строке функции вызывается метод `fetch`. Он тоже возвращает промис, поэтому уже здесь можно применить оператор `await`.

Затем полученные публикации мы парсим с помощью метода `json()`. Вновь применяем оператор `await`, а после этого возвращаем результат. Как работает `fetch` повторяться не будем — про него рассказали в отдельном разделе.

Чтобы воспользоваться функцией `getPosts`, мы опять применяем `await`. Затем выводим список публикаций. Обратите внимание, сначала в консоль будет выведена фраза «Список публикаций», а только потом массив с публикациями.

Обработка ошибок

Применять `async`/`await` несложно. Код действительно становится более лаконичным и простым, но как быть с обработкой ошибок? При использовании промисов мы применяли метод `catch` или передавали отдельный колбэк для обработки ошибки вторым параметром в `then`. Как эти задачи решать при использовании этих операторов?

Для обработки ошибок нам потребуется воспользоваться операторами `try/catch`. Взгляните на пример:

```
// Специально объявим некорректный адрес
const URL = 'https://jsonplaceholder.typicode';

const getPosts = async () => {
    let response;
    try {
        response = await fetch(URL);
    } catch (error) {
        console.log(error);
        return [];
    }

  const posts = await response.json();
```

```
  return posts;
};

(async () => {
    const posts = await getPosts()
    console.log('Список публикаций: ');
    console.log(posts);
})();
```

Пример делает то же самое, что и предыдущий, но в него мы добавили обработку ошибки с помощью блока `try/catch`. Чтобы воспроизвести ошибку мы специально указали некорректный адрес до сервиса. При выполнении метода `fetch` возникнет ошибка, которая будет перехвачена в блоке `catch`. Содержимое ошибки выводится в консоль, а затем из функции возвращается пустой массив. Попробуйте запустить этот код в консоли.

Promise.all и await

Мы уже знаем, что `await` помогает дождаться разрешения промиса, а как быть если требуется подождать несколько промисов? Из раздела «Промисы» мы знаем: для этой задачи у объекта `Promise` есть метод `all`. Он принимает массив промисов. Этот же метод может применяться совместно с `await`. Стоит помнить, что вызов должен происходить внутри `async` функции.

Резюме

`async/await` прекрасная возможность писать асинхронный код в синхронном стиле. Начните применять операторы `async/await` постепенно. Помните, в основе лежат промисы, поэтому крайне важно сначала разобраться с ними, а потом переходить к применению синтаксического сахара.

Прочитали главу?

Нажмите кнопку «Готово», чтобы сохранить прогресс.

Готово

Поделитесь, как вам глава учебника?