**Асинхронное программирование: от колбэков к async/await**

Асинхронное программирование — это основа работы с операциями, которые выполняются в течение неопределенного времени, например, загрузка данных с сервера, чтение файлов или запросы к API. В JavaScript асинхронность реализована так, чтобы основной поток программы оставался свободным и не блокировался длительными операциями. Эта лекция познакомит вас с основами асинхронного программирования, начиная с колбэков и завершая использованием современного синтаксиса async/await.

**Асинхронность в JavaScript**

JavaScript является однопоточным языком, что означает выполнение кода в одном потоке. Однако за счет событийного цикла (Event Loop) язык может эффективно управлять задачами, которые требуют времени на выполнение, например сетевые запросы. Асинхронные операции передаются в фоновую очередь, освобождая основной поток для выполнения других задач.

Для реализации асинхронности в JavaScript исторически использовались три подхода:

1. **Колбэки**
2. **Промисы**
3. **async/await**

**Колбэки**

Колбэки — это функции, которые передаются в другую функцию для выполнения после завершения асинхронной операции. Однако подход с колбэками имеет недостатки: код становится сложным для понимания, особенно при последовательных вызовах, что называется "адом колбэков".

Пример использования колбэков:

function fetchData(callback) {

    setTimeout(() => {

      callback("Данные загружены");

    }, 1000);

  }

  fetchData((message) => {

    console.log(message); // Вывод: Данные загружены

  });

**Промисы**

Для решения проблемы вложенности колбэков в ES6 был введен объект Promise. Промисы упрощают работу с асинхронным кодом, позволяя использовать цепочки методов then и catch для обработки успешного выполнения или ошибок.

const fetchData = () => {

    return new Promise((resolve, reject) => {

      setTimeout(() => {

        const success = true;

        if (success) {

          resolve("Данные загружены");

        } else {

          reject("Ошибка загрузки");

        }

      }, 1000);

    });

  };

  fetchData()

    .then((message) => console.log(message)) // Вывод: Данные загружены

    .catch((error) => console.error(error));

**Async/Await**

С введением async/await в ES2017 (ES8) работа с асинхронным кодом стала еще проще и интуитивно понятной. Ключевое слово async перед функцией позволяет использовать внутри нее await, которое временно приостанавливает выполнение функции до завершения промиса. Это делает код более линейным и читаемым.

Ключевые особенности:

* async превращает функцию в асинхронную, автоматически возвращающую промис.
* await заставляет код ждать завершения промиса и возвращает его результат.
* const fetchData = () => {
* return new Promise((resolve, reject) => {
* setTimeout(() => {
* const success = true;
* if (success) {
* resolve("Данные загружены");
* } else {
* reject("Ошибка загрузки");
* }
* }, 1000);
* });
* };
* const loadData = async () => {
* try {
* const message = await fetchData(); // Ждем завершения промиса
* console.log(message);             // Вывод: Данные загружены
* } catch (error) {
* console.error(error);             // Обработка ошибок
* }
* };
* loadData();