Замыкания в JavaScript

Основные понятия

Замыкание — это функция вместе со всеми внешними переменными, которые ей доступны. В JavaScript функции сохраняют доступ к переменным из того лексического окружения, где они были созданы.

Лексическое окружение (Lexical Environment)

* Специальный внутренний объект
* Содержит локальные переменные и параметры функции
* Имеет ссылку на внешнее окружение ([[Scope]])

Как работают замыкания

1. При создании функции она получает скрытое свойство [[Scope]]
2. Это свойство ссылается на лексическое окружение, где функция была создана
3. При вызове функции создается новое лексическое окружение
4. Для поиска переменных используется цепочка лексических окружений

Практические примеры

1. Простой счетчик

javascript

Copy

Download

function makeCounter() {

let count = 0;

return function() {

return count++;

};

}

const counter = makeCounter();

console.log(counter()); // 0

console.log(counter()); // 1

console.log(counter()); // 2

2. Независимые счетчики

javascript

Copy

Download

const counter1 = makeCounter();

const counter2 = makeCounter();

console.log(counter1()); // 0

console.log(counter1()); // 1

console.log(counter2()); // 0 (независимый счет)

3. Решение проблемы с циклами и событиями

javascript

Copy

Download

// Проблема (все кнопки выводят 10)

for (var i = 0; i < 10; i++) {

button.addEventListener('click', function() {

console.log(i); // Всегда 10

});

}

// Решение с замыканием

for (var i = 0; i < 10; i++) {

(function(x) {

button.addEventListener('click', function() {

console.log(x); // Правильный индекс

});

})(i);

}

Применение замыканий

1. **Создание приватных переменных**:

javascript

Copy

Download

function createUser(name) {

let privateName = name;

return {

getName() { return privateName; },

setName(newName) { privateName = newName; }

};

}

const user = createUser('Анна');

console.log(user.getName()); // "Анна"

1. **Мемоизация (кеширование)**:

javascript

Copy

Download

function memoize(fn) {

const cache = {};

return function(arg) {

return cache[arg] || (cache[arg] = fn(arg));

};

}

1. **Каррирование функций**:

javascript

Copy

Download

function multiply(a) {

return function(b) {

return a \* b;

};

}

const double = multiply(2);

console.log(double(5)); // 10

Важные особенности

1. Замыкания могут приводить к утечкам памяти, если неосторожно хранить ссылки на большие объекты
2. В современных версиях JavaScript (ES6+) можно использовать let/const в циклах вместо IIFE
3. Замыкания активно используются в модульных системах и паттернах проектирования

Замыкания — мощный инструмент JavaScript, позволяющий создавать изолированные контексты и сохранять состояние между вызовами функций.