**[Декоратор-шпион](https://learn.javascript.ru/call-apply-decorators" \l "dekorator-shpion)**

важность: 5

Создайте декоратор spy(func), который должен возвращать обёртку, которая сохраняет все вызовы функции в своём свойстве calls.

Каждый вызов должен сохраняться как массив аргументов.

Например:

function work(a, b) {

alert( a + b ); // произвольная функция или метод

}

work = spy(work);

work(1, 2); // 3

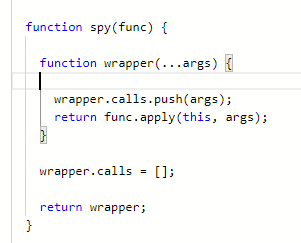
work(4, 5); // 9

for (let args of work.calls) {

alert( 'call:' + args.join() ); // "call:1,2", "call:4,5"

}

P.S.: Этот декоратор иногда полезен для юнит-тестирования. Его расширенная форма – sinon.spy – содержится в библиотеке [Sinon.JS](https://sinonjs.org/).



**[Задерживающий декоратор](https://learn.javascript.ru/call-apply-decorators" \l "zaderzhivayuschiy-dekorator)**

важность: 5

Создайте декоратор delay(f, ms), который задерживает каждый вызов f на ms миллисекунд. Например:

function f(x) {

alert(x);

}

// создаём обёртки

let f1000 = delay(f, 1000);

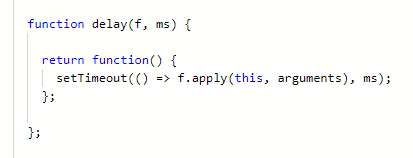
let f1500 = delay(f, 1500);

f1000("test"); // показывает "test" после 1000 мс

f1500("test"); // показывает "test" после 1500 мс

Другими словами, delay(f, ms) возвращает вариант f с «задержкой на ms мс».

В приведённом выше коде f – функция с одним аргументом, но ваше решение должно передавать все аргументы и контекст this.



**[Декоратор debounce](https://learn.javascript.ru/call-apply-decorators" \l "dekorator-debounce)**

важность: 5

Результатом декоратора debounce(f, ms) должна быть обёртка, которая передаёт вызов f не более одного раза в ms миллисекунд. Другими словами, когда мы вызываем debounce, это гарантирует, что все остальные вызовы будут игнорироваться в течение ms.

Например:

let f = debounce(alert, 1000);

f(1); // выполняется немедленно

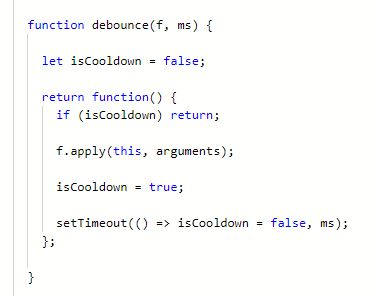
f(2); // проигнорирован

setTimeout( () => f(3), 100); // проигнорирован (прошло только 100 мс)

setTimeout( () => f(4), 1100); // выполняется

setTimeout( () => f(5), 1500); // проигнорирован (прошло только 400 мс от последнего вызова)

На практике debounce полезен для функций, которые получают/обновляют данные, и мы знаем, что повторный вызов в течение короткого промежутка времени не даст ничего нового. Так что лучше не тратить на него ресурсы.



### [Тормозящий (throttling) декоратор](https://learn.javascript.ru/call-apply-decorators" \l "tormozyaschiy-throttling-dekorator)

важность: 5

Создайте «тормозящий» декоратор throttle(f, ms), который возвращает обёртку, передавая вызов в f не более одного раза в ms миллисекунд. Те вызовы, которые попадают в период «торможения», игнорируются.

**Отличие от debounce – если проигнорированный вызов является последним во время «задержки», то он выполняется в конце.**

Давайте рассмотрим реальное применение, чтобы лучше понять это требование и выяснить, откуда оно взято.

**Например, мы хотим отслеживать движения мыши.**

В браузере мы можем объявить функцию, которая будет запускаться при каждом движении указателя и получать его местоположение. Во время активного использования мыши эта функция запускается очень часто, это может происходить около 100 раз в секунду (каждые 10 мс).

**Мы бы хотели обновлять информацию на странице при передвижениях.**

…Но функция обновления update() слишком ресурсоёмкая, чтобы делать это при каждом микродвижении. Да и нет смысла делать обновление чаще, чем один раз в 1000 мс.

Поэтому мы обернём вызов в декоратор: будем использовать throttle(update, 1000) как функцию, которая будет запускаться при каждом перемещении указателя вместо оригинальной update(). Декоратор будет вызываться часто, но передавать вызов в update() максимум раз в 1000 мс.

Визуально это будет выглядеть вот так:

1. Для первого движения указателя декорированный вариант сразу передаёт вызов в update. Это важно, т.к. пользователь сразу видит нашу реакцию на его перемещение.
2. Затем, когда указатель продолжает движение, в течение 1000 мс ничего не происходит. Декорированный вариант игнорирует вызовы.
3. По истечению 1000 мс происходит ещё один вызов update с последними координатами.
4. Затем, наконец, указатель где-то останавливается. Декорированный вариант ждёт, пока не истечёт 1000 мс, и затем вызывает update с последними координатами. В итоге окончательные координаты указателя тоже обработаны.

Пример кода:

function f(a) {

console.log(a)

}

// f1000 передаёт вызовы f максимум раз в 1000 мс

let f1000 = throttle(f, 1000);

f1000(1); // показывает 1

f1000(2); // (ограничение, 1000 мс ещё нет)

f1000(3); // (ограничение, 1000 мс ещё нет)

// когда 1000 мс истекли ...

// ...выводим 3, промежуточное значение 2 было проигнорировано

P.S. Аргументы и контекст this, переданные в f1000, должны быть переданы в оригинальную f.

