JavaScript

Урок №15: Основы.Стрелочные функции.

Автор Бахшиллоев Бехзод
Ментор по Frontend в PRO UNITY

Введение.

Функции-стрелки (стрелочные функции, функции-ракеты) - новый синтаксис для создания функциональных выражений.

Особенности arrow function:
1) более лаконичный синтаксис;
2) отсутствие псевдомассива аргументов arguments;
3) лексическое определение this;
4) не могут использоваться в качестве конструкторов (с оператором new); **это мы не проходили**
5) не могут использоваться для создания генераторов. **это мы не проходили**

Arrow function всегда создается с помощью **function expression и не имеет имени (анонимная)**. Для создания arrow functions не используется ключевое слово function и вместо этого используется fat arrow (жирная стрелка) **=>**

Объявление.

```
const foo= () => { return 2 + 3; } →
// function foo() { return 2 + 3; }

const boo = () => {
  const number = Math.random();  return number;
} →
/* function boo() {
  const number = Math.random(); return number;
} */
```

Если тело функции содержит однострочное выражение, то фигурные скобки можно опустить, и слово return в данном случае тоже опускается:

```
const test = () => 2 + 3;
// function test() { return 2 + 3; }
```

Однако если в функции более одной строки, то следует использовать традиционный синтаксис:

```
const test = () => {
  const pi = Math.PI;
  return pi * 2;
}
```

Объявление.

Если функция принимает один аргумент, то круглые скобки вокруг аргумента можно не использовать:

```
const getModule = (number) => { return Math.abs(number); }
→
const getModule = number => { return Math.abs(number); }
→
const getModule = number => Math.abs(number);
```

Однако при отсутствии аргументов или наличии более одного аргументов, круглые скобки обязательны.

Если стрелочная функция используется без фигурных скобок и слова return, и при этом мы хотим вернуть объект, то объект нужно заключить в круглые скобки:

```
const getTextInfo = (text) => {
  return { length: text.length, isEven: !(text.length % 2) };
}
getTextInfo('ola!'); // {length: 4, isEven: true}

const getTextInfo = text => ({ length: text.length, isEven: !(text.length % 2) });
getTextInfo('hello'); // {length: 5, isEven: false}
```

Примеры.

function *getText*() { return 'Hello!'; }	1) **const** getText = **()** => { return 'Hello!'; } 2) **const** getText = **()** => 'Hello!';

function *getGreet*(name) {
  return 'Hello, ' + name;
}
	1) **const** getGreet = **(**name**)** => {
	  return 'Hello, ' + name;
	}
	
	2) **const** getGreet = **(**name**)** => 'Hello, ' + name;
	
	3) **const** getGreet = name => {
	  return 'Hello, ' + name;
	}
	
	4) **const** getGreet = name => 'Hello, ' + name;

Функции стрелки как callback.

Функции-стрелки очень удобно использовать в качестве коллбэков, например, при переборе массива. Сравните:

```
[0,1,2,3,4,5,6,7].filter(function (number) { return number % 2});
// [1, 3, 5, 7]


[0,1,2,3,4,5,6,7].filter(number => number % 2);
// [1, 3, 5, 7]


[0,1,2,3].map(number => ({ digit: number}));
// [ {digit: 0}, {digit: 1}, {digit: 2} ...]

[0,1,2,3,4,5,6,7].sort((prev, next) => prev - next);

document.body.addEventListener('click', (e) => {// some actions});
```

Функции стрелки. Arguments.

Внутри обычных функций можно обратиться к псевдомассиву arguments, который содержит список параметров, с которыми была вызвана функция. Внутри arrow function такая возможность отсутствует.

Проблему отсутствия arguments в стрелочных функциях можно решить путём сочетания функций-стрелок с rest оператором:

```
const printParams = (...props) => console.log(props);
printParams('test', 123); // ["test", 123]
```

В данном случае переменная props будет истинным массивом, так как rest оператор всегда возвращает нативный массив (не псевдомассив).

Функции стрелки. this.

При создании функций традиционным синтаксисом ключевое слово this определяется в момент вызова функции и зависит от способа вызова функции.

Для стрелочных функций не существует “собственного” this, то есть они используют this “выше” уровнем - **тот this, который определяется в момент создания актуального окружения, в котором находится стрелочная функция**.

```
const user = { age: 45,
                    getAge: () => { return this.age; }
                };
user.getAge(); // undefined
```

В данном случае объект user создавался в глобальном пространстве, следовательно, в момент создания объекта this являлся глобальным объектом (Window).

Задачи:

1. Переделать функцию с использованием функции-стрелки (в методе reduce тоже использовать arrow function):

```
function sum() {
  const params = Array.prototype.slice.call(arguments);

  if (!params.length) return 0;

  return params.reduce(function (prev, next) { return prev + next; });
}

sum(1, 2, 3, 4); // 10
sum(); // 0
```

что такое метод reduce можно прочитать здесь