

Южно-Уральский  
государственный  
университет

Национальный  
исследовательский  
университет

# КРОК

## Основы JS. Функции обратного вызова. Запросы к серверу

КРОК

Челябинск, ул. Карла Маркса, д. 38

Смирнов Анатолий  
Технический менеджер

Кузнецов Сергей  
Старший инженер-разработчик

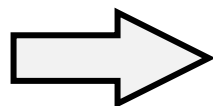
Фоменко Алексей  
Младший инженер-разработчик



- Зачастую конструкция `function()` может занимать много места, в ES6 ввели новое объявление функций в стиле «лямба-выражений»

```
(param1, param2, ..., paramN) => {  
    statements  
}
```

```
function f(a1, a2) {  
    console.log(a1 + a2);  
}
```

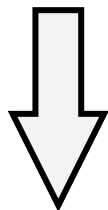


```
const f = (a1, a2) => {  
    console.log(a1 + a2);  
}
```



- Если возвращаемое значение короткое, то можно упразднить {}:

```
(param1, param2, ..., paramN) =>  
{  
    return expression;  
}
```

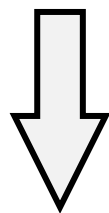


```
(param1, param2, ..., paramN) => expression
```

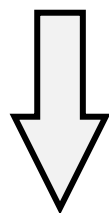


- Пример объявления функции без фигурных скобок:

```
function f(a1, a2) {  
    console.log(a1 + a2);  
}
```



```
const f = (a1, a2) => {  
    console.log(a1 + a2);  
}
```



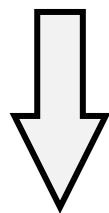
```
const f = (a1, a2) => console.log(a1 + a2);
```

```
> const f = (a1, a2) => console.log(a1 + a2);  
← undefined  
> f(1, 2)  
3
```

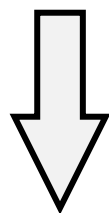


- Пример объявления функции без фигурных скобок:

```
function f (x, y) {  
    return x * y;  
}
```



```
const f = (x, y) => {  
    return x * y;  
}
```



```
const f = (x) => x * x;
```



- Фигурные скобки нужны, если тело функции требует нескольких операций или для повышения читаемости:

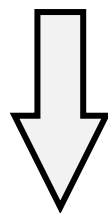
```
const f = (a1, a2) => {  
  if (a1 > 0) {  
    console.warn(a1 + a2);  
  }  
  
  const s = `Текст ${a1} и ${a2}`;  
  return s;  
}  
  
const result = f(1, 2);
```

```
> const f = (a1, a2) => {  
  if (a1 > 0) {  
    console.warn(a1 + a2);  
  }  
  
  const s = `Текст ${a1} и ${a2}`;  
  return s;  
}  
  
const result = f(1, 2);  
▲ ▶ 3  
◀ undefined  
> result  
◀ 'Текст 1 и 2'
```



- Если аргументов нет, то при объявлении должны быть просто пустые скобки ():

```
function f() {  
    console.log(123);  
}
```



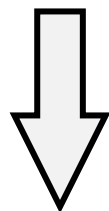
```
const f = () => console.log(123);
```

```
> const f = () => console.log(123);  
↵ undefined  
> f()  
123
```

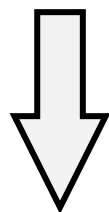


- Если аргумент один, то круглые скобки можно упразднить:

```
function f(a1) {  
    console.log(a1);  
}
```



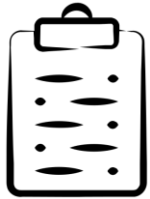
```
const f = (a1) => console.log(a1);
```



```
const f = a1 => console.log(a1);
```

```
> const f = a1 => console.log(a1);  
← undefined  
> f('Hello')  
Hello
```





- Аналогичные по функционалу конструкции:

```
[1, 2, 3].forEach(function(item) {  
    console.log(item * 2)  
})
```

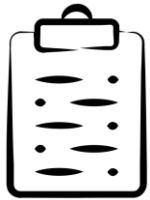
```
[1, 2, 3].forEach((item) => {  
    console.log(item * 2)  
})
```

```
[1, 2, 3].forEach(item => {  
    console.log(item * 2)  
})
```

```
[1, 2, 3].forEach(item => console.log(item * 2));
```



- Очень распространённой задачей в современных веб-сайтах и приложениях является получение отдельных элементов данных с сервера для обновления разделов веб-страницы без необходимости загрузки всей новой страницы
- JavaScript может отправлять сетевые запросы на сервер и подгружать новую информацию по мере необходимости



- Fetch API предоставляет интерфейс JavaScript для работы с запросами и ответами HTTP.
- Он также предоставляет глобальный метод `fetch()`

```
fetch('http://example.com', {  
  // Опции  
})  
  .then(function(response) {  
    return response.json();  
  })  
  .then(function(data) {  
    console.log(data);  
  });
```

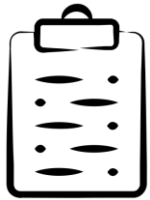


Для представления ответа от сервера в Fetch API применяется интерфейс Response. Свойства:

- ok: хранит булево значение, которое указывает, завершился ли запрос
- status: хранит статусный код ответа
- statusText: хранит сообщение статуса, которое соответствует статусному коду
- ...

```
fetch('http://example.com')  
  .then(function(response) {  
    console.log(response)  
  })
```

```
▼ Response {type: 'basic', url: 'http://loca  
  ► body: ReadableStream  
    bodyUsed: true  
  ► headers: Headers {}  
    ok: true  
    redirected: false  
    status: 200  
    statusText: "OK"  
    type: "basic"  
    url: "http://localhost/render/api.php"
```



Методы:

- `text()` для текста
- `json()` для текста в формате JSON
- `blob()` для бинарных объектов (изображения, аудио и т.д.)
- ...

```
fetch('http://example.com')  
  .then(function(response) {  
    return response.json();  
  })  
  .then(function(data) {  
    // data – объект из JSON  
  });
```



Для установки метода используется свойство `method` в опциях `fetch`. По умолчанию GET

```
fetch('https://httpbin.org/put', {  
  method: 'PUT'  
})  
.then(function(response) {  
  return response.json()  
})  
.then(function(data) {  
  console.log(data)  
})
```

```
▼ {args: {...}, data: '', files: {...}, form: {...}, headers: {...},  
  ▶ args: {}  
    data: ""  
    ▶ files: {}  
    ▶ form: {}  
    ▶ headers: {Accept: '*/*', Accept-Encoding: 'gzip, deflate'  
      json: null  
      origin: "88.206.74.184"  
      url: "https://httpbin.org/put"  
    }  
    ▶ [[Prototype]]: Object
```



Для отправки тела используется свойство `body` в опциях `fetch`

```
let data = {
  key1: 'value1'
}
fetch('https://httpbin.org/post', {
  method: 'POST',
  body: JSON.stringify(data)
})
.then(function(response) {
  return response.json()
})
.then(function(data) {
  console.log(data)
})
```

```
▼ {args: {...}, data: '{"key1":"value1"}', files: {...}, form: {...}}
  ► args: {}
  data: "{\\"key1\\":\\"value1\\"}"
  ► files: {}
  ► form: {}
  ► headers: {Accept: '*/*', Accept-Encoding: 'gzip, deflate,'}
  ► json: {key1: 'value1'}
  origin: "88.206.74.184"
  url: "https://httpbin.org/post"
  ► [[Prototype]]: Object
```



Для установки заголовков используется свойство `headers` в опциях `fetch`

```
let data = {
  key1: 'value1'
}
fetch('https://httpbin.org/post', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
.then(function(response) {
  return response.json()
})
.then(function(data) {
  console.log(data)
})
```

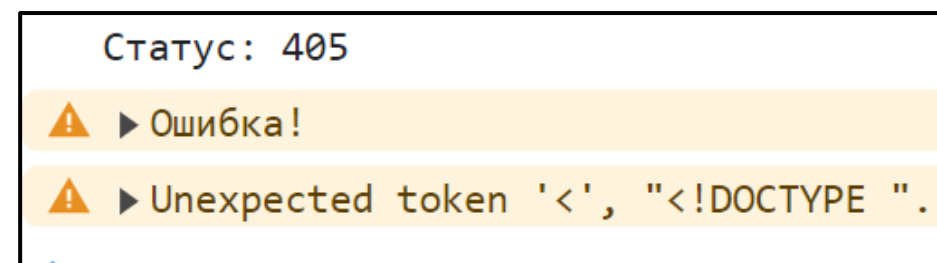
```
▼ {args: {...}, data: '{"key1":"value1"}', files: {...}, form: {...}
  ► args: {}
  data: "{\\"key1\\":\\"value1\\"}"
  ► files: {}
  ► form: {}
  ▼ headers:
    Accept: "*/*"
    Accept-Encoding: "gzip, deflate, br"
    Accept-Language: "ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7"
    Content-Length: "17"
    Content-Type: "application/json"
    Host: "httpbin.org"
    Origin: "http://localhost"
```





Для обработки ошибок применяется метод catch

```
fetch('https://httpbin.org/post', { // post
  method: 'PUT' // put
})
.then(function(response) {
  console.log('Статус: ' + response.status)
  return response.json()
})
.then(function(data) {
  console.log(data)
})
.catch(function(error) { // Обработка ошибок
  console.warn('Ошибка!')
  console.warn(error.message)
});
```



```
fetch('https://httpbin.org/post', { // post
  method: 'PUT' // put
})
.then(function(response) {
  console.log('Статус: ' + response.status)
  if (response.status === 200) {
    return response.json()
  }
  return response.text()
})
.then(function(data) {
  console.log(data)
})
.catch(function(error) {
  console.warn('Ошибка!')
  console.warn(error.message)
});
```

```
fetch('https://httpbin.org/post', { // post
  method: 'PUT' // put
})
.then(function(response) {
  console.log('Статус: ' + response.status)
  if (response.status === 200) {
    return response.json()
  }
  return response.text()
})
.then(function(data) {
  console.log(data)
})
.catch(function(error) {
  console.warn('Ошибка!')
  console.warn(error.message)
});
```

Статус: 405

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
```

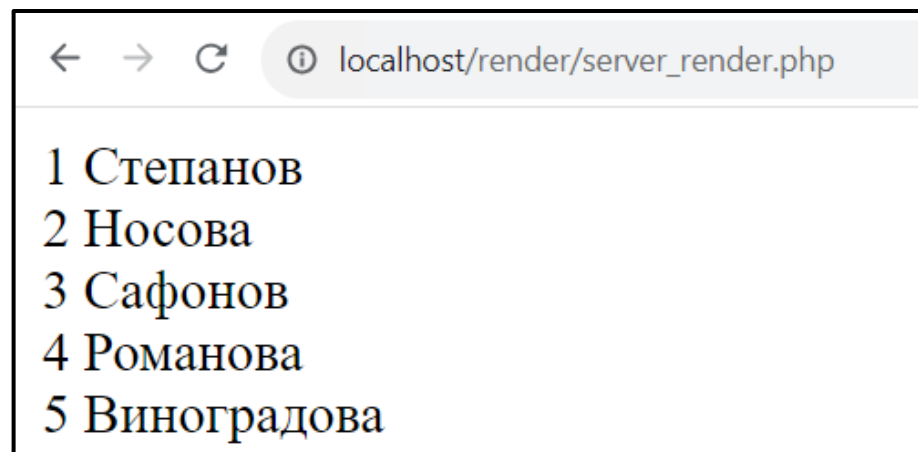


- Серверный рендеринг генерирует полный HTML страницы на сервере в ответ на навигацию. Это позволяет избежать дополнительных проходов для получения данных и шаблонов на клиенте, так как это выполняется до того, как браузер получает ответ
- Рендеринг на стороне клиента (CSR) означает рендеринг страниц непосредственно в браузере с использованием JavaScript. Вся логика, сбор данных, построение DOM-модели и маршрутизация обрабатываются на клиенте, а не на сервере

```
<?php
    $dsn = "pgsql:host=localhost;port=5432;dbname=postgres";
    $pdo = new PDO($dsn, 'postgres', '1');

    $sql = "select id, surname from users limit 5";
    $stmt = $pdo->query($sql);

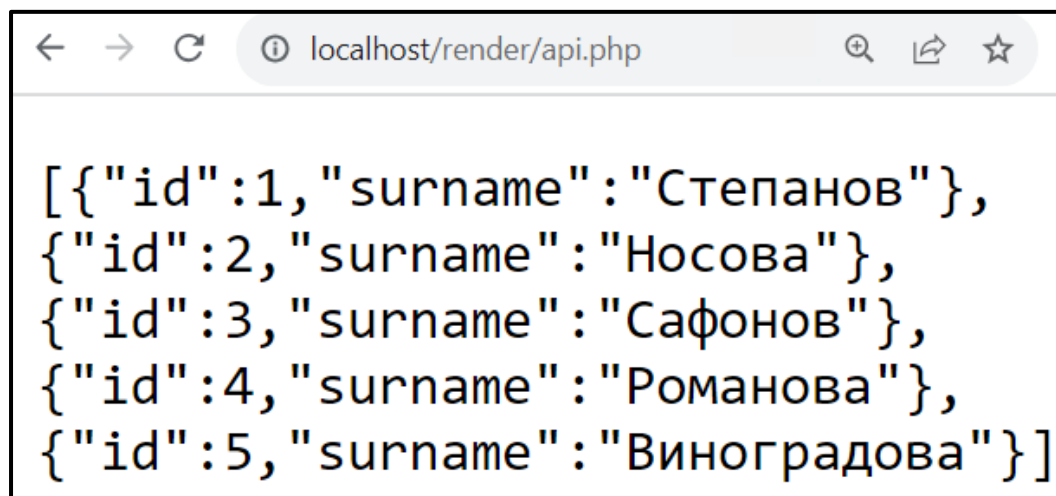
    while ($row = $stmt->fetch()) {
        echo $row['id']. " " . $row['surname'] . "<br/>";
    }
?>
```



```
<?php
    $dsn = "pgsql:host=localhost;port=5432;dbname=postgres;";
    $pdo = new PDO($dsn, 'postgres', '1');

    $sql = "select id, surname from users limit 5";
    $stmt = $pdo->query($sql);

    $result_list = $stmt->fetchAll(PDO::FETCH_ASSOC);
    header('Content-Type: application/json');
    echo json_encode($result_list, JSON_UNESCAPED_UNICODE);
?>
```



The screenshot shows a web browser window with the address bar displaying 'localhost/render/api.php'. The main content area shows a JSON array of five objects, each containing an 'id' and a 'surname'. The surnames are in Cyrillic: Степанов, Носова, Сафионов, Романова, and Виноградова.

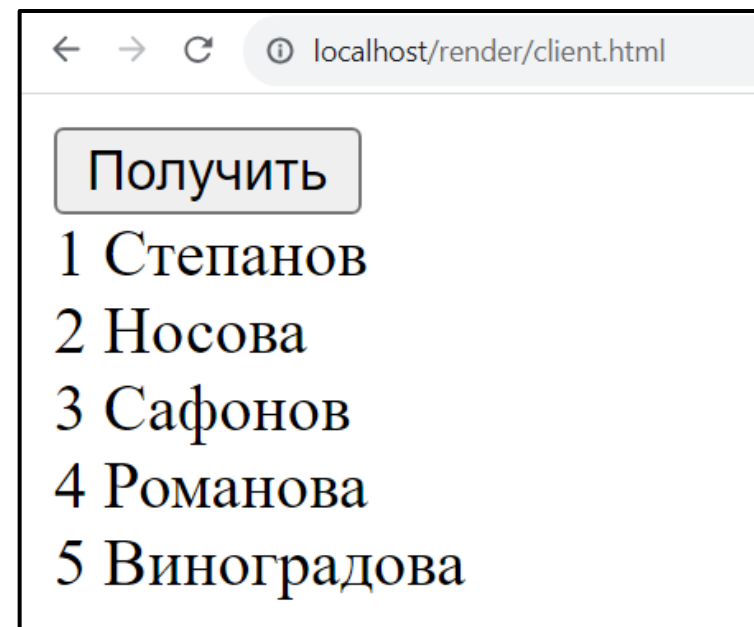
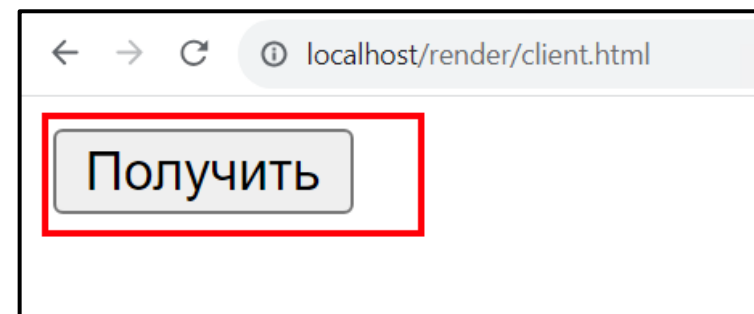
```
[{"id":1,"surname":"Степанов"},
{"id":2,"surname":"Носова"},
{"id":3,"surname":"Сафионов"},
{"id":4,"surname":"Романова"},
{"id":5,"surname":"Виноградова"}]
```

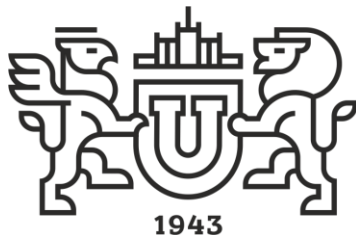
```
<meta charset="UTF-8">
<button id="sendButton" onclick="getUsers()">Получить</button>
<div id="container"></div>
```

```
<script>
const cont = document.querySelector('#container');
```

```
function onSuccess(data) {
  cont.innerHTML = '';
  for (let row of data) {
    cont.innerHTML += row['id'] + ' ' +
                      row['surname'] + '<br/>';
  }
}
```

```
function getUsers() {
  fetch('http://localhost/render/api.php')
    .then(function (result) {
      return result.json();
    })
    .then(onSuccess)
}
</script>
```





Южно-Уральский  
государственный  
университет

Национальный  
исследовательский  
университет

# КРОК

## Спасибо за внимание!



КРОК

Челябинск, ул. Карла Маркса, д. 38