



Handlebars. IF, Циклы, данные в шаблоне

Урок №12

Теория



8 минут



**Кто решил задачу с передачей
данных в шаблон?**

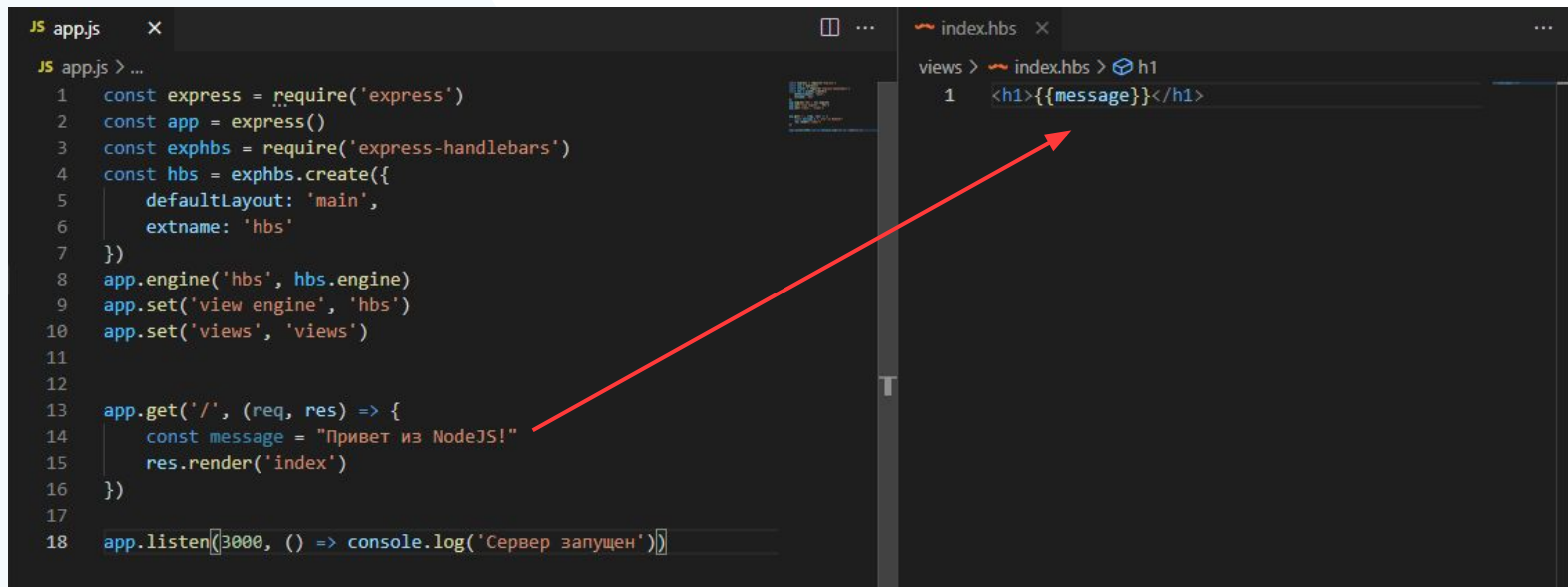




ПЕРЕДАЧА ДАННЫХ В ШАБЛОН

На стороне NodeJS мы можем что-то посчитать, создать и т.д., после чего передать в шаблонизатор, который это встроит в разметку.

Передача данных



```
JS app.js x
JS app.js > ...
1 const express = require('express')
2 const app = express()
3 const exphbs = require('express-handlebars')
4 const hbs = exphbs.create({
5   defaultLayout: 'main',
6   extname: 'hbs'
7 })
8 app.engine('hbs', hbs.engine)
9 app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get('/', (req, res) => {
14   const message = "Привет из NodeJS!"
15   res.render('index')
16 })
17
18 app.listen(3000, () => console.log('Сервер запущен'))
```

```
index.hbs x
views > index.hbs > h1
1 <h1>{{message}}</h1>
```

**Но как это сделать без чтения
? файла и замены текста?**

Очень просто!

Передача данных

```
1. app.get('/', (req, res) => {  
2.   const message = "Привет из NodeJS!"  
3.   res.render('index', {})  
4. })
```



Передаем объект с данными
вторым аргументом в метод render!

Передача данных

```
1. app.get('/', (req, res) => {  
2.   const message = "Привет из NodeJS!"  
3.   res.render('index', {  
4.     message: message  
5.   })  
6. })
```



Каждое свойство объекта
будет доступно в шаблоне!

Передача данных

JS app.js

JS app.js > ...

```
1  const express = require('express')
2  const app = express()
3  const exphbs = require('express-handlebars')
4  const hbs = exphbs.create({
5    |   defaultLayout: 'main',
6    |   extname: 'hbs'
7  })
8  app.engine('hbs', hbs.engine)
9  app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get('/', (req, res) => {
14   const message = "Привет из NodeJS!"
15   res.render('index', {
16     |   message: message
17   })
18 })
19
20 app.listen(3000, () => console.log('Сервер запущен'))
```

Передаем наше
сообщение!

index.hbs

views > index.hbs > h1

```
1  <h1>{{message}}</h1>
```



Синтаксис

После передачи объекта в шаблон, любое поле можно отображать при помощи двух фигурных скобок и названия свойства объекта.

Передача данных

JS app.js

JS app.js > ...

```
1  const express = require('express')
2  const app = express()
3  const exphbs = require('express-handlebars')
4  const hbs = exphbs.create({
5    |   defaultLayout: 'main',
6    |   extname: 'hbs'
7  })
8  app.engine('hbs', hbs.engine)
9  app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get('/', (req, res) => {
14   const message = "Привет из NodeJS!"
15   res.render('index', {
16     |   message: message
17   })
18 })
19
20 app.listen(3000, () => console.log('Сервер запущен'))
```

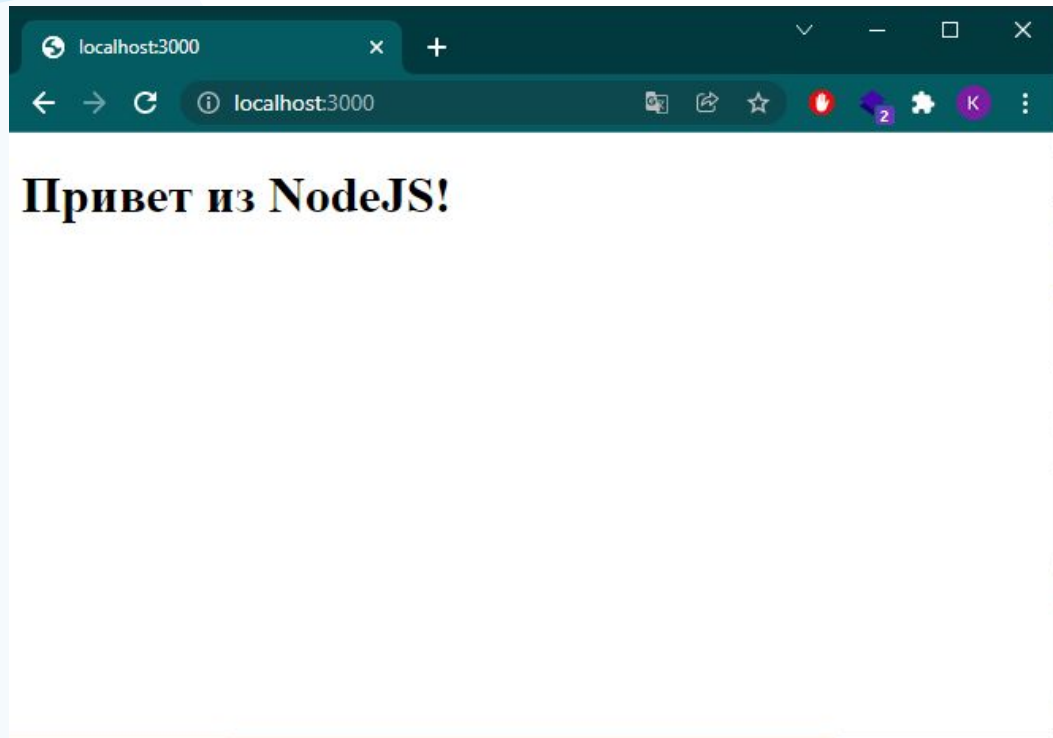
index.hbs

views > index.hbs > h1

1 <h1>{{message}}</h1>

Встраиваем
значение свойства в
шаблон!

Данные встроились в шаблон!





Какие есть вопросы, друзья?

Практика



12 минут





Теория



15 минут





Директива IF

В зависимости от переданных данных можно выбирать, будет ли отображаться тот или иной контент.

Быть или не быть

```
1. app.get('/', (req, res) => {  
2.   res.render('index', {  
3.     name: "Иван",  
4.     lastname: "Иванов",  
5.     visible: true  
6.   })  
7. })
```



Boolean значение!

При работе с директивой IF есть правила

Его можно использовать если значение свойства:

- ⬡ boolean (true/false)
- ⬡ undefined
- ⬡ null
- ⬡ ""
- ⬡ 0
- ⬡ []

**Нам недоступны привычные
операторы сравнения**

При работе с директивой IF есть правила

Нельзя использовать это:

⬡ ==, ===

⬡ <, >

⬡ <=, >=

⬡ !

**Значение свойства неявно
приводится к Bool**

Синтаксис

1. `{{#if visible}}` ← Начало IF
2. `<h1>Текст</h1>`
3. `{{/if}}` ← Конец IF

Синтаксис

1. `{{#if visible}}`
2. `<h1>Текст</h1>`
3. `{{/if}}`

Если условие окажется истинным, то всё, что между `#if` и `/if` отобразится на странице!



Синтаксис

1. {{#if visible == true}}

2. <h1>Текст</h1>

3. {{/if}}

4.

5. {{#if !visible}}

6. <h1>Текст</h1>

7. {{/if}}

Вызовет ошибку!



Синтаксис

```
1.  {{#if visible}}  
2.    <h1>Текст 1</h1>  
3.  {{else}}  
4.    <h1>Текст 2</h1>  
5.  {{/if}}
```

Можно добавить `else` и то, что
написано после него
отобразится в случае, если
условие окажется ложным!

Синтаксис

1. `{{#if visible}}`
2. `<h1>Текст 1</h1>`
3. `{{else}}`
4. `<h1>Текст 2</h1>`
5. `{{/if}}`

В данном случае, если значение свойства `visible` окажется `true`, в шаблон будет встроен тег с Текст 1, иначе будет встроен тег с Текст 2



А если нужно встроить в шаблон что-либо, если значение переменной false?

Так как мы не имеем возможности использовать операторы сравнения при проверке значения, нужно пользоваться другой директивой, которая добавит в шаблон что-либо, если значения свойства FALSE!

Обратная проверка

1. `{{#unless visible}}`
2. `<h1>Текст</h1>`
3. `{{/unless}}`

В данном случае, если значение свойства `visible` окажется `false`, в шаблон будет встроен тег. Эту директиву можно воспринимать как обратную директиве IF

Ну, а пока... вопросыки?

Практика



15 минут



Теория



15 минут



Работа с массивами

```
1. app.get('/', (req, res) => {  
2.   res.render('index', {  
3.     names: ["Иван", "Николай", "Алексей"]  
4.   })  
5. })
```



Как вывести элементы массива
на страницу?

Что произойдет?

```
JS app.js x
JS app.js > ...
1  const express = require('express')
2  const app = express()
3  const exp_hbs = require('express-handlebars')
4  const hbs = exp_hbs.create({
5    |   defaultLayout: 'main',
6    |   extname: 'hbs'
7  })
8  app.engine('hbs', hbs.engine)
9  app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get("/", (req, res) => {
14   |   res.render("index", {
15   |   |   names: ["Иван", "Николай", "Алексей"]
16   |   })
17 })
18
19
20 app.listen(3000, () => console.log('Сервер запущен'))
```

```
views > index.hbs > h1
1  <h1>{{names}}</h1>
```

Переданный массив со строками!

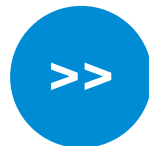
Что произойдет?



- Ошибка шаблонизации



- Выведет тип данных

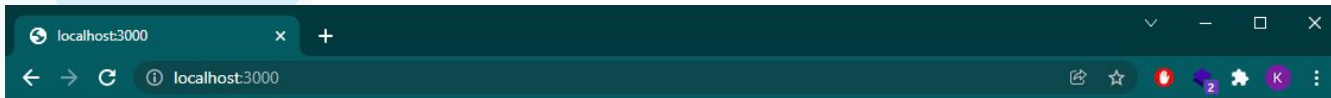


- Ничего не выведет



- Все элементы массива

Все элементы массива



Иван, Николай, Алексей

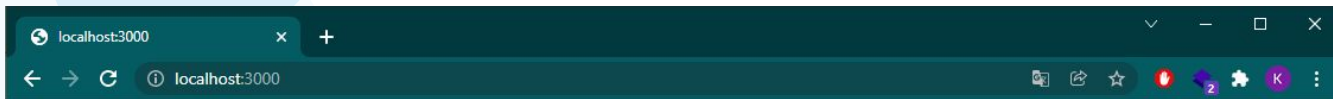
Массив объектов

```
JS app.js x
JS app.js > ...
1 const express = require('express')
2 const app = express()
3 const exphbs = require('express-handlebars')
4 const hbs = exphbs.create({
5   defaultLayout: 'main',
6   extname: 'hbs'
7 })
8 app.engine('hbs', hbs.engine)
9 app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get("/", (req, res) => {
14   res.render("index", {
15     names: [
16       { name: 'Иван', lastname: 'Иванов' },
17       { name: 'Николай', lastname: 'Николаев' },
18       { name: 'Алексей', lastname: 'Алексеев' }
19     ]
20   })
21 })
22
23
24 app.listen(3000, () => console.log('Сервер запущен'))
25
```

```
index.hbs x
views > index.hbs > h1
1 <h1>{{names}}</h1>
```

Переданный массив объектов!

Все элементы массива



[object Object],[object Object],[object Object]



Цикл

Тут нам и понадобится цикл, чтобы получать значения свойств объектов и корректно их выводить. На самом деле, если мы захотим создать теги ``, содержимое которых будет элементами массива, всё равно без цикла не обойтись.

Синтаксис

1. `{{#each names}}` ← Цикл по массиву `names`
2. `<p>{{ this }}</p>`
3. `{{/each}}`

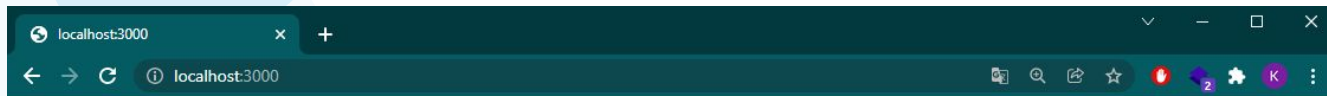
Синтаксис

1. `{{#each names}}`

2. `<p>{{ this }}</p>`  **this - итерируемый элемент!**

3. `{{/each}}`

Массив объектов



[object Object]

[object Object]

[object Object]

**А как получать значения
свойств?**

**Просто написать их название
вместо this!**

Цикл по массиву объектов

```
JS app.js x
JS app.js > ...
1  const express = require('express')
2  const app = express()
3  const exphbs = require('express-handlebars')
4  const hbs = exphbs.create({
5    defaultLayout: 'main',
6    extname: 'hbs'
7  })
8  app.engine('hbs', hbs.engine)
9  app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get("/", (req, res) => {
14   res.render("index", {
15     names: [
16       { name: 'Иван', lastname: 'Иванов' },
17       { name: 'Николай', lastname: 'Николаев' },
18       { name: 'Алексей', lastname: 'Алексеев' }
19     ]
20   })
21 })
22
23
24 app.listen(3000, () => console.log('Сервер запущен'))
25
```

```
index.hbs x
views > index.hbs > ...
1  {{#each names}}
2    <p>{{name}} {{lastname}}</p>
3  {{/each}}
4
5
```

Названия свойств объекта!

Цикл по массиву объектов

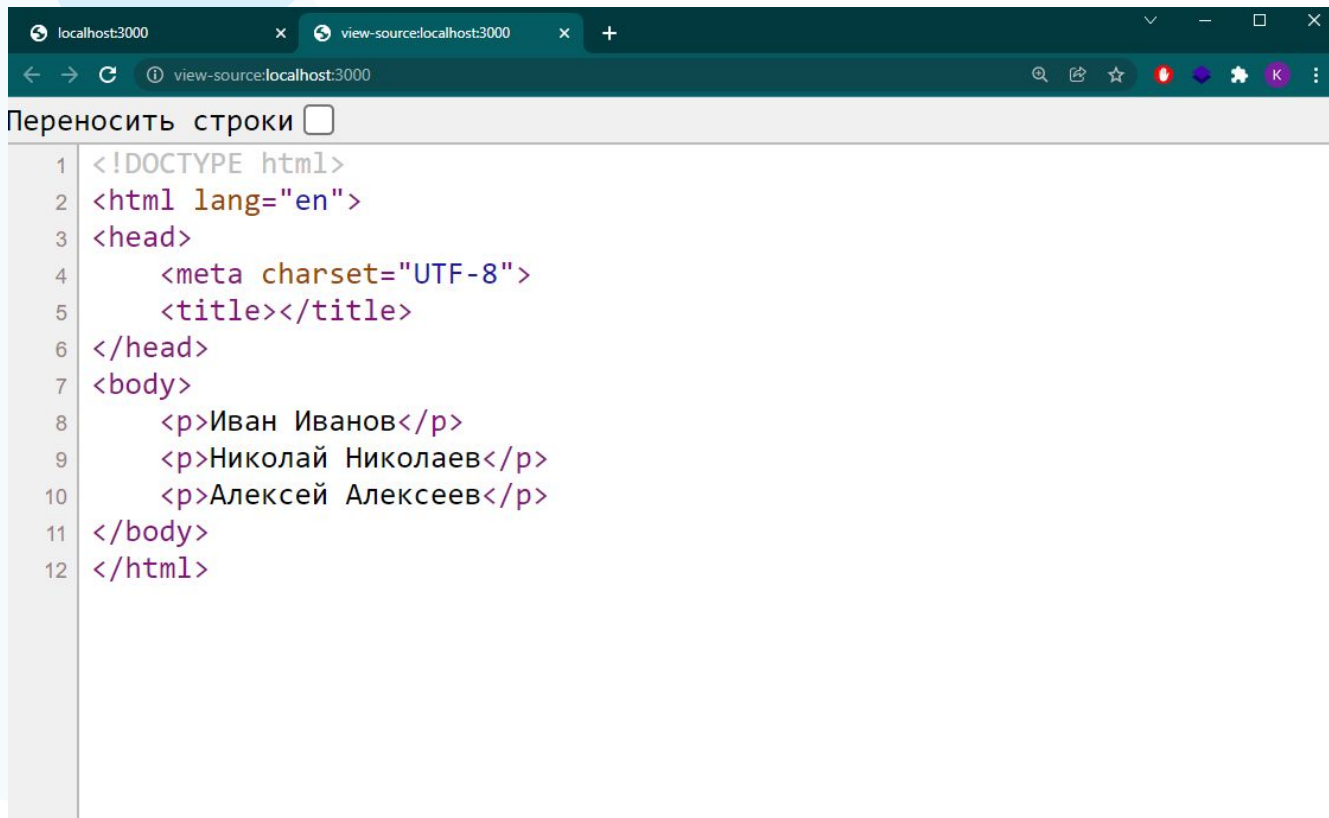


Иван Иванов

Николай Николаев

Алексей Алексеев

Цикл по массиву объектов



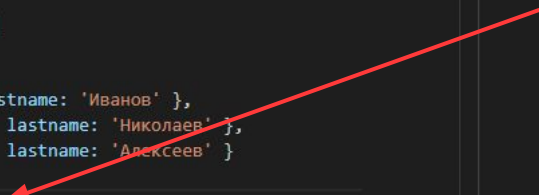
```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title></title>
6 </head>
7 <body>
8   <p>Иван Иванов</p>
9   <p>Николай Николаев</p>
10  <p>Алексей Алексеев</p>
11 </body>
12 </html>
```

Что выведется?

```
JS app.js x index.hbs x
JS app.js > app.get("/") callback > name
1 const express = require('express')
2 const app = express()
3 const exphbs = require('express-handlebars')
4 const hbs = exphbs.create({
5   defaultLayout: 'main',
6   extname: '.hbs'
7 })
8 app.engine('hbs', hbs.engine)
9 app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get("/", (req, res) => {
14   res.render("index", {
15     names: [
16       { name: 'Иван', lastname: 'Иванов' },
17       { name: 'Николай', lastname: 'Николаев' },
18       { name: 'Алексей', lastname: 'Алексеев' }
19     ],
20     name: 'Кирилл!!!!!!'
21   })
22 })
23
24
25 app.listen(3000, () => console.log('Сервер запущен'))
26
```

views > index.hbs > ...

```
1 {{#each names}}
2   <p>{{name}} {{lastname}}</p>
3 {{/each}}
4
5
```



Значение свойства объекта!

**А как вывести значения
одноименного свойства в
цикле, не относящееся к
итерируемому объекту?**

@root.name



@root

@root - зона видимости данных, не относящаяся к конкретному итерируемому элементу. Такое написание предполагает, что мы ищем нужное свойство из глобальной зоны видимости.

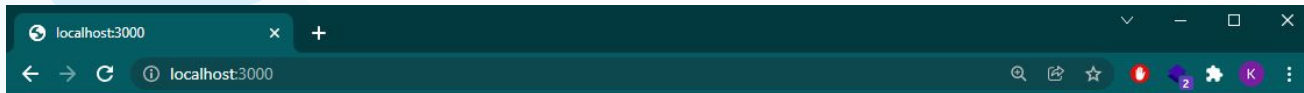
Глобальная зона видимости

```
JS app.js  X
JS app.js > app.get("/") callback > name
1  const express = require('express')
2  const app = express()
3  const exp_hbs = require('express-handlebars')
4  const hbs = exp_hbs.create({
5    defaultLayout: 'main',
6    extname: 'hbs'
7  })
8  app.engine('hbs', hbs.engine)
9  app.set('view engine', 'hbs')
10 app.set('views', 'views')
11
12
13 app.get("/", (req, res) => {
14   res.render("index", {
15     names: [
16       { name: 'Иван', lastname: 'Иванов' },
17       { name: 'Николай', lastname: 'Николаев' },
18       { name: 'Алексей', lastname: 'Алексеев' }
19     ],
20     name: 'Кирилл!!!!!!'
21   })
22 })
23
24
25 app.listen(3000, () => console.log('Сервер запущен'))
26
```

```
index.hbs  X
views > index.hbs > ...
1  {{#each names}}
2    <p>{{@root.name}} {{lastname}}</p>
3  {{/each}}
4
5
```

Глобальное свойство name

Глобальная зона видимости



Кирилл!!!!!! Иванов

Кирилл!!!!!! Николаев

Кирилл!!!!!! Алексеев



@root

Написание @root требуется в том случае, если в цикле необходимо выводить данные из глобальной зоны видимости, не относящиеся к итерируемым объектам!

Какие есть вопросы?

Практика



20 минут



Итоги урока

- 1) Научились передавать данные в шаблон
- 2) Познакомились с директивой if/else
- 3) Узнали про обратную директиву unless
- 4) Научились генерировать разметку в цикле
- 5) Изучили работу Handlebars