

петък, 30.06.2017

Events & Forms

Синтаксисът тук е подобен на този в html , само че е **camelCase**, тоест **onclick** в **html** става **onClick** в **react** Всички javascript-ски евънти(събития) , които познаваме съществуват и тук , но се пишат в **camelCase**

Обикновено за да използваме дадена функция, която искаме да закачим към даден **event**(събитие), ние прикачваме метод от компонента. *Това е най-лесният вариант , когато JSX-а (html кодът) се намира в същия компонент.*

КОГАТО ИЗПОЛЗВАМЕ КЛАСОВЕ, ЗАДЪЛЖИТЕЛНО ТРЯБВА ДА БАЙНДНЕМ(bind-нем) this-а. ИНАЧЕ НЯМА ДА РАБОТИ

Даден метод може да се bind-не или при прикачане на евента , примерно **<h1 onClick={this.headerClick.bind(this)}>Click me!</h1>** или в конструктора както е показано в снимката по-долу.

```
index.js  JS App.js  x
1  import React, { Component } from 'react'
2  import './App.css'
3
4  class App extends Component {
5    constructor (props) {
6      super(props)
7
8      this.headerClick = this.headerClick.bind(this)
9    }
10
11    headerClick (event) {
12      window.alert('CLICKED')
13      console.log(event)
14      console.log(event.target)
15    }
16
17    render () {
18      return (
19        <div className='App'>
20          <h1 onClick={this.headerClick.bind(this)}>Click me!</h1>
21        </div>
22      )
23    }
24  }
25
26  export default App
```

Няма значение кой от двата варианта се прави.. важното е кодът да е консистентен, тоест навсякъде да се прави само по единия начин.

Ако имаме **<input type='text' />** ,// поведението по подразбиране(по default) на реакт е , че по никакъв начин не можем да хванем нещата за този **input**// . *Тоест нищо не можем да правим с него*

Тук задължително трябва да направим **onChange**, за да прихванем стойностите.

event.target е самият обект, върху който е кликнато.

Как можем да вземем информация кой точно **input** се е сменил ?

1. **Създаване на обект, който да съдържа информация за всички input елементи**

let field = target.name -> С това вземаме кои точно input ни се е променил
let value = target.value -> С това вземаме неговата стойност

Друго което се използва често е да подаваме към някакви **child** елементи някаква информация от сорта на **events**(събития)

```
class Child extends Component {
  render () {
    return (
      <div>
        <a onClick={this.props.clickMe}>Click me</a>
        <input type='text' name='someInput' onChange={this.props.changeMe} />
      </div>
    )
  }
}
```

Примерно както е показано на горната картинка, имаме някакъв child component , който има input , name и някакви други неща, НО не той решава каква да е логиката.

На следващата снимка е показано как в child елементи чрез props се подават функции, които обработват събития

```
4
5 class App extends Component {
6   constructor (props) {
7     super(props)
8
9     this.handleClick = this.handleClick.bind(this)
10    this.inputChange = this.inputChange.bind(this)
11  }
12
13  handleClick (event) {
14    window.alert('CLICKED')
15    console.log(event)
16    console.log(event.target)
17  }
18
19  inputChange (event) {
20    const target = event.target
21    let field = target.name
22    let value = target.value
23
24    console.log(`${field} - ${value}`)
25  }
26
27  render () {
28    return (
29      <div className='App'>
30        <Child handleClick={this.handleClick.bind(this)} inputChange={this.inputChange.bind(this)} />
31      </div>
32    )
33  }
34 }
35
36 export default App
37
```

Forms (Форми)

Формите са си нормална форма, като единственото нещо, което трябва да се добави е какво да прави при **onChange** и при събитие на формата.

Тоест на всеки `<input />` в нашата форма трябва да закачим **onChange** event

НАЙ-ДОБРИ ПРАКТИКИ

- **винаги формата като форма е отделен компонент**
- **опитваме се и най-елементарните input-и да ги извадим ако започнат да се повтарят.**
- **винаги искаме да имаме client-side validation. Какво означава това ? - да се валидира примерно някакъв input преди да бъде пратен на сървъра. Този вид заявки предотвратява излишни заявки към сървъра.**
- **друго нещо, което е добра практика да се прави е да имаме преизползвами(reusable) компоненти(components) дори и за много елементарни неща като `<input />`**

Две неща са важни за формата.

```
JS index.js JS App.js JS AuthorPage.js x JS Child.js
1 import React, { Component } from 'react'
2
3 class CreateAuthorPage extends Component {
4   render () {
5     return (
6       <div>
7         <h1>Create Author:</h1>
8         <form>
9           <label htmlFor='firstName'>First Name:</label>
10          <input type='text' name='firstName' id='firstName' /><br />
11          <label htmlFor='lastName'>Last Name:</label>
12          <input type='text' name='lastName' id='lastName' /><br />
13          <input type='submit' value='Add Author' />
14        </form>
15      </div>
16    )
17  }
18 }
19
20 export default CreateAuthorPage
21
```

```
JS index.js x JS App.js x JS AuthorPage.js JS Child.js
1 import React, { Component } from 'react'
2 import './App.css'
3 import CreateAuthorPage from './components/AuthorPage'
```

```

4
5 class App extends Component {
6   render() {
7     return (
8       <div className='App'>
9         <CreateAuthorPage />
10      </div>
11    )
12  }
13 }
14
15 export default App
16

```

Две неща са важни за формата:

- формичката да я опишем, нагласим както е показано на следващата снимка:

```

JS index.js JS App.js JS EditAuthorPage.js JS AuthorForm.js JS CreateAuthorPage.js x
25 saveAuthor (event) {
26   event.preventDefault()
27   console.log(this.state.author)
28 }
29
30 render() {
31   return (
32     <div>
33       <h1>Create Author:</h1>
34       <form>
35         <label htmlFor='firstName'>First Name:</label>
36         <input
37           type='text'
38           name='firstName'
39           id='firstName'
40           onChange={this.handleChange.bind(this)}
41           value={this.state.author.firstName} /><br />
42         <label htmlFor='lastName'>Last Name:</label>
43         <input
44           type='text'
45           name='lastName'
46           id='lastName'
47           onChange={this.handleChange.bind(this)}
48           value={this.state.author.lastName} /><br />
49         <input type='submit' value='Add Author' onClick={this.saveAuthor.bind(this)} />
50       </form>
51     </div>
52   )
53 }
54

```

- След това да закачим *onChange* event (в случая се нарича *handleChange*) и *save* event. Показано е на следващата картинка

JS index.js

JS App.js

JS EditAuthorPage.js

JS AuthorForm.js

JS CreateAuthorPage.js x

```
14
15  handleInputChanged (event) {
16    const target = event.target
17    const name = target.name
18    const value = target.value
19
20    let author = this.state.author
21    author[name] = value
22    this.setState({author})
23  }
24
25  saveAuthor (event) {
26    event.preventDefault()
27    console.log(this.state.author)
28  }
29
30  render () {
31    return (
32      <div>
33        <h1>Create Author:</h1>
34        <form>
35          <label htmlFor='firstName'>First Name:</label>
36          <input
37            type='text'
38            name='firstName'
39            id='firstName'
40            onChange={this.handleInputChanged.bind(this)}
41            value={this.state.author.firstName} /><br />
42          <label htmlFor='lastName'>Last Name:</label>
43          <input
44            type='text'
45            name='lastName'
46            id='lastName'
47            onChange={this.handleInputChanged.bind(this)}
48            value={this.state.author.lastName} /><br />
49          <input type='submit' value='Add Author' onClick={this.saveAuthor.bind(this)} />
50        </form>
51      </div>
52    )
53  }
54 }
```

Ln 31, Col 13 Spaces: 2 UTF-8