Сейчас мы с вами начнем изучать работу с формами в фреймворке React. Начнем с простых инпутов и будем постепенно разбирать более сложные элементы форм.

Основы работы с формами

Пусть у нас в **this.state.value** хранится текст 'привет':

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}
}
```

Теперь в методе **render** сделаем инпут, пока просто выведем его на экран:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	//Выведем инпут:
	render() {
		return <div>
			<input />
		</div>;
	}
}
```

Обратите внимание на то, что инпут надо закрывать с помощью обратного слеша, иначе при преобразовании JSX будет ошибка.

Давайте теперь сделаем так, чтобы в **value** инпута записалось значение из **this.state.value**:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	render() {
		return <div>
			<input value={this.state.value} />
		</div>;
	}
```

```
}
```

Запустите этот код - на экране вы увидите инпут с текстом 'привет'. Однако, вас ждет сюрприз: *вы не сможете поменять текст нашего инпута*. Попробуйте сами - повводите что-нибудь в этот инпут - текст просто не будет меняться.

Почему так? Потому что мы четко сказали, что в **value** инпута должно быть значение из **this.state.value**. Это значение не меняется - и значит **value** инпута тоже не будет меняться, даже если вы вручную что-то попытаетесь туда написать.

Понятно, что такое поведение не очень удобно и нам нужно что-то с этим сделать. Что именно: нужно организовать *двухстороннее связывание* - **this.state.value** и **value** инпута должны зависеть друг от друга: при изменении одного должен меняться и другой.

Первый шаг для этого следующий: нужно к нашему инпуту добавить событие **onChange** и привязать к нему какой-нибудь метод, назовем его, к примеру, **handleChange**.

Сделаем это:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	//Срабатывает при любом изменении инпута:
	handleChange() {
		//тут какой-то код
	}

	render() {
		return <div>
			<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		</div>;
	}
}
```

Как работает событие **onChange** - оно срабатывает при попытке любого изменения инпута. Например, если мы пытаемся ввести в него какой-то текст, то **onChange** будет срабатывать при каждом вводе символа.

И, хотя текст инпута не будет меняться из-за привязанного **this.state.value**, событие **onChange** будет срабатывать и каждый раз вызывать метод **handleChange**.

Убедимся в этом - сделаем так, чтобы **handleChange** при каждом своем вызове выводил что-нибудь в браузер. Запустите следующий код, повводите что-нибудь в инпут и посмотрите в консоль - вы увидите, как при каждом вводе символа вызывается **handleChange**:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	//Выводим текст '!!!' каждый раз при изменении инпута:
	handleChange() {
		console.log('!!!');
	}

	render() {
		return <div>
			<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		</div>;
	}
}
```

Давайте теперь сделаем так, чтобы содержимое инпута могло изменяться. Для этого в методе **handleChange** нужно в свойство **this.state.value** записывать содержимое атрибута **value** нашего инпута.

Однако, нас ждет некоторая неожиданность - мы не сможем получить **value** инпута так, как мы привыкли: такое - **this.value** - не будет работать, ведь **this** сейчас привязан с помощью **bind** к объекту нашего класса **App** и не указывает на инпут, в котором происходит событие.

Но как тогда обратиться к нашему инпуту? Ответ такой: с помощью объекта Event, вот так:

```
handleChange(event) {
	console.log(event.target); //наш инпут;
	console.log(event.target.value) //значение нашего инпута;
}
```

То есть: на наш инпут ссылается **event.target**, а значение инпута можно достать из **event.target.value**.

Вооружившись этим знаниями, привяжем теперь значение нашего инпута **event.target.value** к **this.state.value** с помощью метода **this.setState**:

```
handleChange(event) {
	this.setState({value: event.target.value});
}
```

Давайте соберем все вместе и запустим наш код (сделайте это) - теперь данные в инпут можно будет вводить:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	//Изменяем this.state.value при изменении инпута:
	handleChange(event) {
		this.setState({value: event.target.value});
	}

	render() {
		return <div>
			<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		</div>;
	}
}
```

Итак, сейчас любые изменения инпута мгновенно приводят к изменению **this.state.value**. Поэтому, если мы где-нибудь в JSX коде выведем содержимое **this.state.value** - оно *мгновенно будет изменятся при вводе текста в инпут*.

Давайте сделаем над нашим инпутом абзац, в который будем выводить содержимое **this.state.value**, вот так:

```
render() {
	return <div>
		<p>текст инпута: {this.state.value}</p>
		<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
	</div>;
}
```

Добавим это изменение в наш класс:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: 'привет'};
	}

	//Записываем value инпута в this.state.value:
	handleChange(event) {
		this.setState({value: event.target.value});
	}

	render() {
		return <div>
			<p>текст инпута: {this.state.value}</p>
			<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		</div>;
	}
}
```

Запустите этот код, повводите текст в инпут и вы увидите, как он мгновенно меняется в абзаце над нашим инпутом.

Модификация данных на выводе

Данные на выводе в JSX коде можно модифицировать с помощью различных методов. Применим, к примеру, к тексту инпута метод toUpperCase. В этом случае текст будет выводится большими буквами:

```
render() {
	return <div>
		<p>текст инпута: {this.state.value.toUpperCase()}</p>
		<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
	</div>;
}
```

Добавим это изменение в наш класс:

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {value: ''}; //пусть инпут изначально пустой
	}

	handleChange(event) {
		this.setState({value: event.target.value});
```

```
	}

	render() {
		return <div>
			<p>текст инпута: {this.state.value.toUpperCase()}</p>
			<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		</div>;
	}
}
```

Запустите этот код, повводите текст в инпут и вы увидите, как он появляется в абзаце в верхнем регистре.

Отправка формы

Иногда может быть неудобным, что при введении каждого символа в инпут каждый раз что-то меняется на экране.

Во-первых, ввод данных еще не закончен и это может привести к неправильной работе скрипта.

Во-вторых, возможно, после ввода данных мы хотим провести какую-то ресурсоемкую операцию над ними. Очевидно, что не стоит выполнять ее после каждого ввода символа - лучше бы в этом случае дождаться окончания ввода.

Для избежания приведенных выше проблем сделаем следующее: создадим HTML форму с инпутом и кнопкой для отправки:

```
render() {
	return <form>
		<input value={this.state.value}
onChange={this.handleChange.bind(this)} />
		<input type="submit" />
	</form>
}
```

Предполагается, что пользователь нашего сайта будет работать с нашей формой так: сначала он введет данные в инпут и затем нажмет на кнопку отправки. В этом случае в теге **<form>** сгенерируется событие **onsubmit**.

Давайте отловим это событие: напишем форме атрибут **onSubmit** и привяжем к нему метод **handleSubmit**:

```
render() {
```

```
        return <form onSubmit={this.handleSubmit.bind(this)}>
                <input value={this.state.value}
onChange={this.handleChange.bind(this)} />
                <input type="submit" />
        </form>
}
```

Обратите внимание на то, к инпуту все равно привязан метод **handleChange** - это нужно для того, чтобы мы вообще что-то могли писать в этот инпут и данные автоматически заносились в **this.state.value**.

Напишем реализацию метода **handleSubmit**: при попытке отправки формы выведем алертом **value** нашего инпута из **this.state.value**, а затем вызовем event.preventDefault() для отмены отправки формы (если это не сделать - по нажатию на кнопку submit форма отправится и страница перезагрузится):

```
handleSubmit(event) {
        alert(this.state.value); //выводим данные на экран
        event.preventDefault(); //отменяем отправку формы
}
```

Итак, давайте соберем все вместе:

```
class App extends React.Component {
        constructor() {
                super();
                this.state = {value: ''};
        }

        //Записываем value инпута в this.state.value:
        handleChange(event) {
                this.setState({value: event.target.value});
        }

        //Выводим this.state.value при отправке формы:
        handleSubmit(event) {
                alert(this.state.value); //выводим данные на экран
                event.preventDefault(); //отменяем отправку формы
        }

        render() {
                return <form onSubmit={this.handleSubmit.bind(this)}>
                        <input value={this.state.value}
onChange={this.handleChange.bind(this)} />
                        <input type="submit" />
                </form>
        }
}
```

Запустите этот код, введите данные в инпут и нажмите на кнопку отправки - вы увидите алерт с текстом нашего инпута.

Добавление в список

Сейчас мы совместим вывод массивов в виде списка и инпут для добавления новых элементов в этот список.

Давайте вначале вспомним код, с помощью которого мы выводили массив в виде списка **ul**, а также добавляли новый элемент в список с помощью кнопочки. Текст у добавленного элемента был жестко задан и имел значение 'пункт':

```
class App extends React.Component {
	constructor() {
		super();
		this.state = {items: [1, 2, 3, 4, 5]};
	}

	//Добавляем в конец списка новый пункт:
	addItem() {
		this.state.items.push('пункт');
		this.setState({items: this.state.items});
	}

	render() {

		//Формируем набор из тегов li:
		const list = this.state.items.map((item, index) => {
			return <li key={index}>{item}</li>;
		});

		//По нажатию на кнопку вызываем addItem:
		return (
			<div>
				<ul>
					{list}
				</ul>
				<button
onClick={this.addItem.bind(this)}>добавить</button>
			</div>
		);
	}
}
```

Добавим теперь в этот код форму, с помощью которой по нажатию на кнопку данные из инпута будут записываться в конец списка:

```
class App extends React.Component {
```

```
	constructor() {
		super();
		this.state = {items: [1, 2, 3, 4, 5], value: ''};
	}

	//Добавляем в конец списка новый пункт:
	addItem(event) {

		//Добавляем новый элемент в массив:
		this.state.items.push(this.state.value);

		//Применяем изменение:
		this.setState({items: this.state.items});

		//Отменяем отправку формы:
		event.preventDefault();
	}

	//Записываем value инпута в this.state.value:
	handleChange(event) {
		this.setState({value: event.target.value});
	}

	render() {

		//Формируем набор из тегов li:
		const list = this.state.items.map((item, index) => {
			return <li key={index}>{item}</li>;
		});

		//По нажатию на кнопку вызываем addItem:
		return (
			<div>
				<ul>
					{list}
				</ul>

				<form onSubmit={this.addItem.bind(this)}>
					<input
						value={this.state.value}
						onChange={this.handleChange.bind(this)}
					/>
					<input type="submit" />
				</form>
			</div>
		);
	}
}
```

Запустите этот код, введите данные в инпут и нажмите на кнопку отправки - значение из инпута добавится в конец списка.