ОНЛАЙН-ОБРАЗОВАНИЕ

```
let lesson = {
  id:       '09'
  themes:  [
    'Components Lifecycle',
    'state', 'props'
  ],
  date:     '29.03.2018',
  teacher: {
    name:      'Юрий Дворжецкий',
    position: 'Lead Developer’
  }
};
```

Как меня слышно && видно?

Если нет – напишите, если слышите – смайлик в чат.

OTUS

Скажите пару слов о React

1. Попробовали?
2. Понравилось?
3. Подсели? ☺
4. Вопросы? ☺

О вебинаре:

Что сможем делать после вебинара?

- Разрабатывать полноценные компоненты React
- И уже приложения (но небольшие)
- Собственно, и почти всё, не считая роутера, Flux/Redux, сахара и опыта

План

- React Components Lifecycle, Events
- State and Props
- Прочее

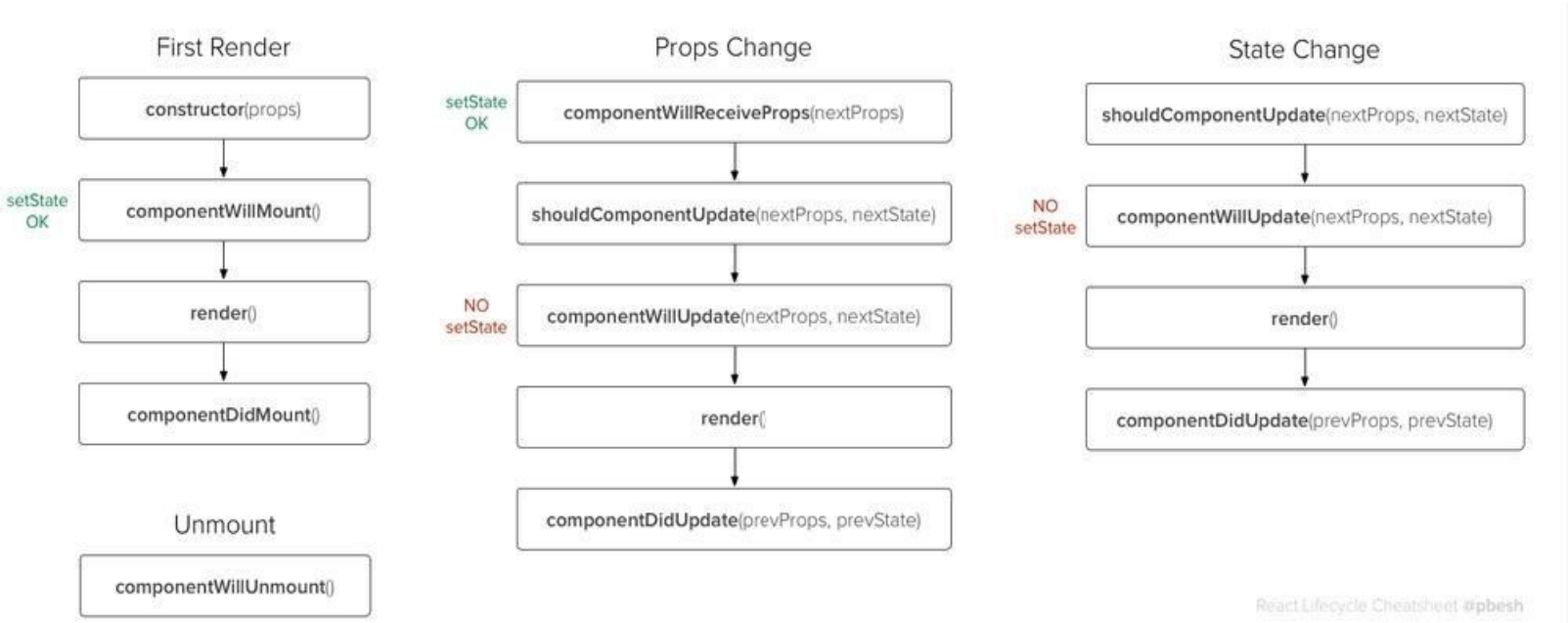
OTUS

React Component Lifecycle

React lifecycle

- Метод render() каждого React компонента, должен быть «чистой функцией» (pure function).
- Это означет, что он должен быть stateless (не менять состояние компонента), не делать никаких Ajax-запросов и т.д.
- Этот метод должен просто принять props и state, а потом просто сгенерировать DOM.
- Так где же это делать?

React lifecycle



constructor

```
class MyComponent extends Component {
    constructor(props) {
        super();
        this.state = {
            // ...
        }
    }
}
```

Конструктор вызывается один раз за всю жизнь объекта.

В конструкторе можно смело задавать начальное состояние.

componentWillMount

```
class MyComponent extends Component {

    componentWillMount() {
        // TODO: add code
    }
}
```

Вызывается ровно один раз за жизнь объекта, перед тем, как произойдёт первый рендер (вызов метода render)

Если Вы вызовете setState внутри этого метода, то state будет корректно обновлён без дополнительного вызова render()

componentWillMount

```
class MyComponent extends Component {

    componentWillMount() {
        // TODO: add code
    }
}
```

Вызывается ровно один раз за жизнь объекта, перед тем, как произойдёт первый рендер (вызов метода render)

Если Вы вызовете setState внутри этого метода, то state будет корректно обновлён без дополнительного вызова render()

render

```
class MyComponent extends Component {

    render() {
        return (
            <span> ! </span>
        )
    }
}
```

В этом методе и возвращается JSX

Вызов метода setState приведёт к повторному вызову render.

Но новый state не будет доступен в этом методе (setState асинхронный)

componentDidMount

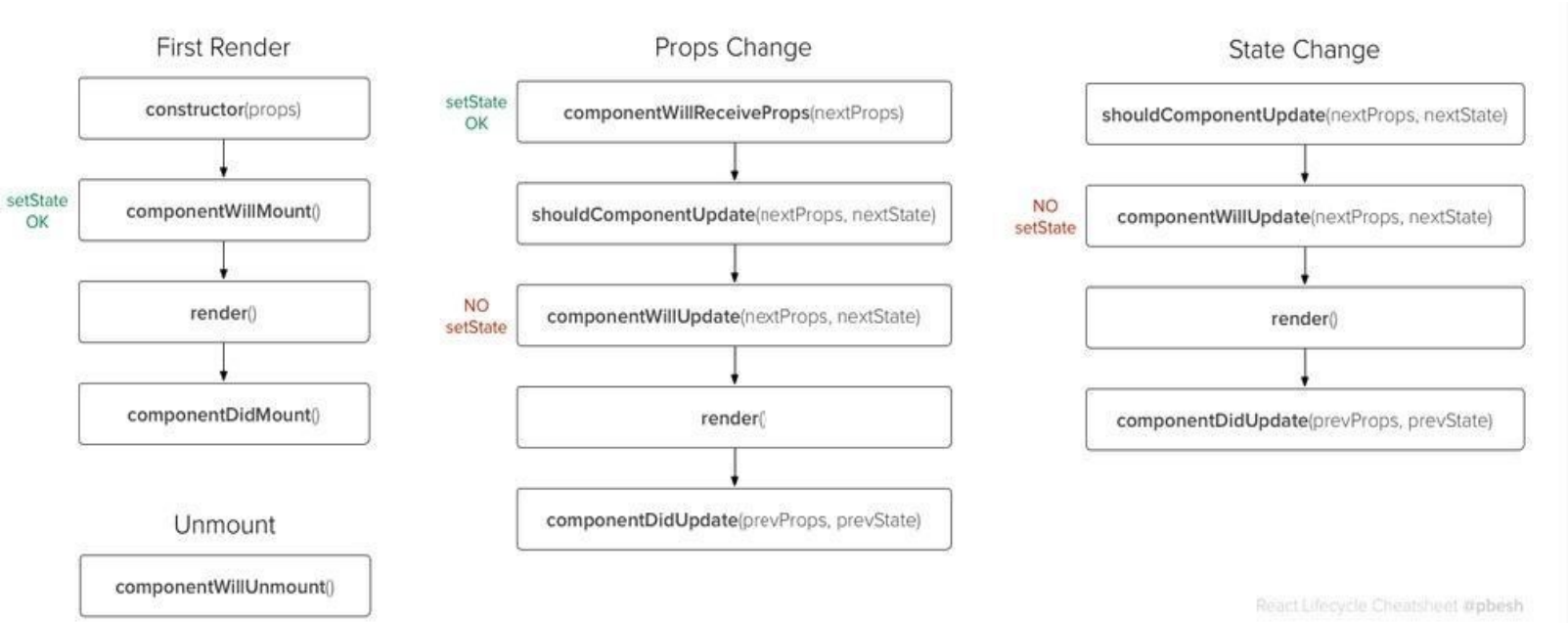
```
class MyComponent extends Component {

    componentDidMount() {
        // TODO: add code
    }
}
```

Вызывается только один раз за жизнь объекта, когда объект привязан к DOM.

В этом методы можно уже получить доступ к DOM или ref (можно добавлять листенров)

Вызывать setState можно, но он приведёт к дополнительном рендеру (такое делают, для красивого открытия страницы).

React lifecycle



componentWillUnmount

```
class MyComponent extends Component {

    componentWillUnmount() {
        // TODO: add code
    }
}
```

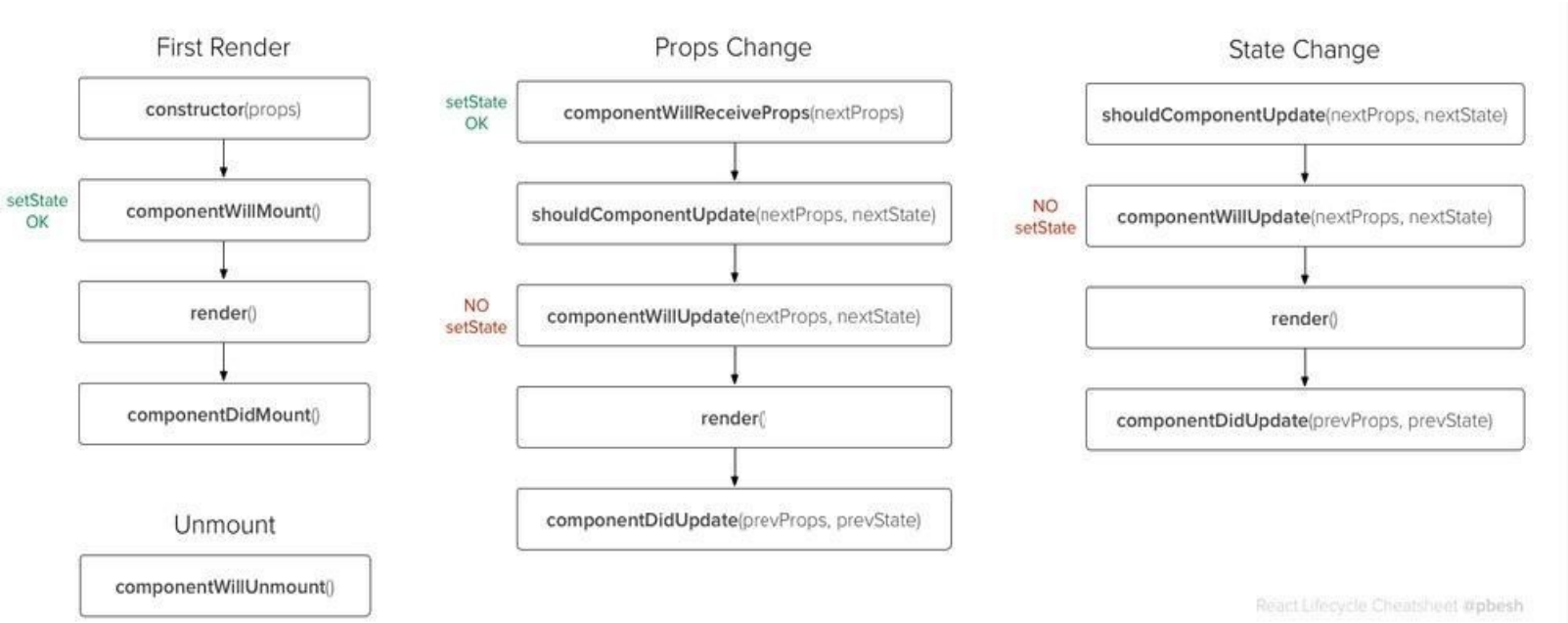
Вызывается только один раз за жизнь объекта, перед тем ка отвязыватеся от DOM.

В этом методы можно ещё получить доступ к DOM или ref

Вызывать setState бесполезно.

Здесь можно отвязывать листенеров.

React lifecycle



componentWillReceiveProps

```
class MyComponent extends Component {

    componentWillReceiveProps(newProps) {
        // TODO: add code
    }
}
```

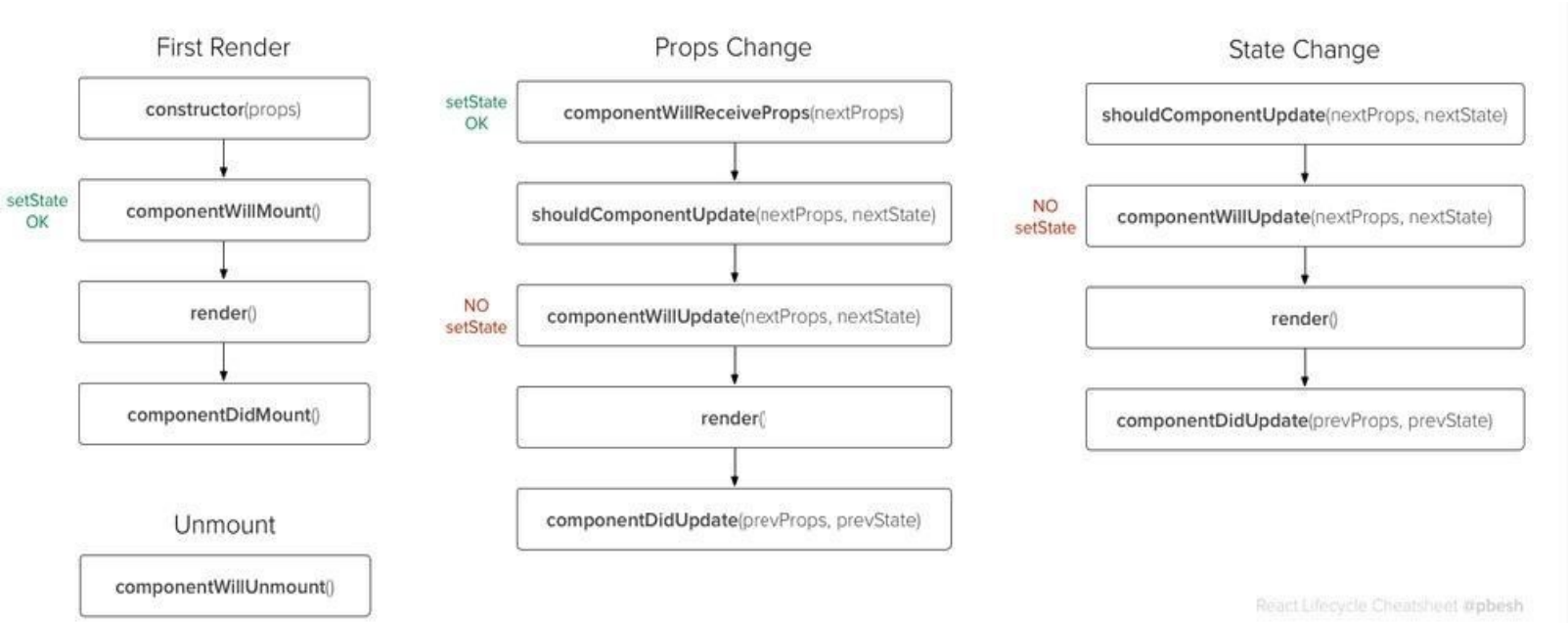
Вызывается каждый раз, как приходят новые свойства

В аргументы приходят новые свойства, старые доступны по this.props

setState в этом методе не приводит к дополнительному рендеру

Здесь state и обновляется в зависимости от новых props

React lifecycle



shouldComponentUpdate

```
class MyComponent extends Component {

    shouldComponentUpdate(nextState, nextProps) {
        // TODO: add code
        return true;
    }
}
```

С помощью этого метода можно сказать React-у, необходимо ли перерисовывать компонент.

shouldComponentUpdate и PureComponent

```
class MyComponent extends PureComponent {..}
```

Собственно, содержит реализацию shouldComponentUpdate

Если объект свойств стал другой (не мутировался, а реально новый, сравнение по ссылке), то только shouldComponentUpdate будет возвращать true

Мутирование props-ов не вызовет re-render, несмотря на изменение содержимого

componentWillUpdate

```
class MyComponent extends Component {

    componentWillUpdate() {
        // TODO: add code
    }
}
```

Вызывается каждый раз перед обновлением DOM компонента, кроме первого рендера.

Все изменения state в этом методе будут переписаны React-ом

Крайне не рекомендуется вообще вызывать setState в этом методе

componentDidUpdate

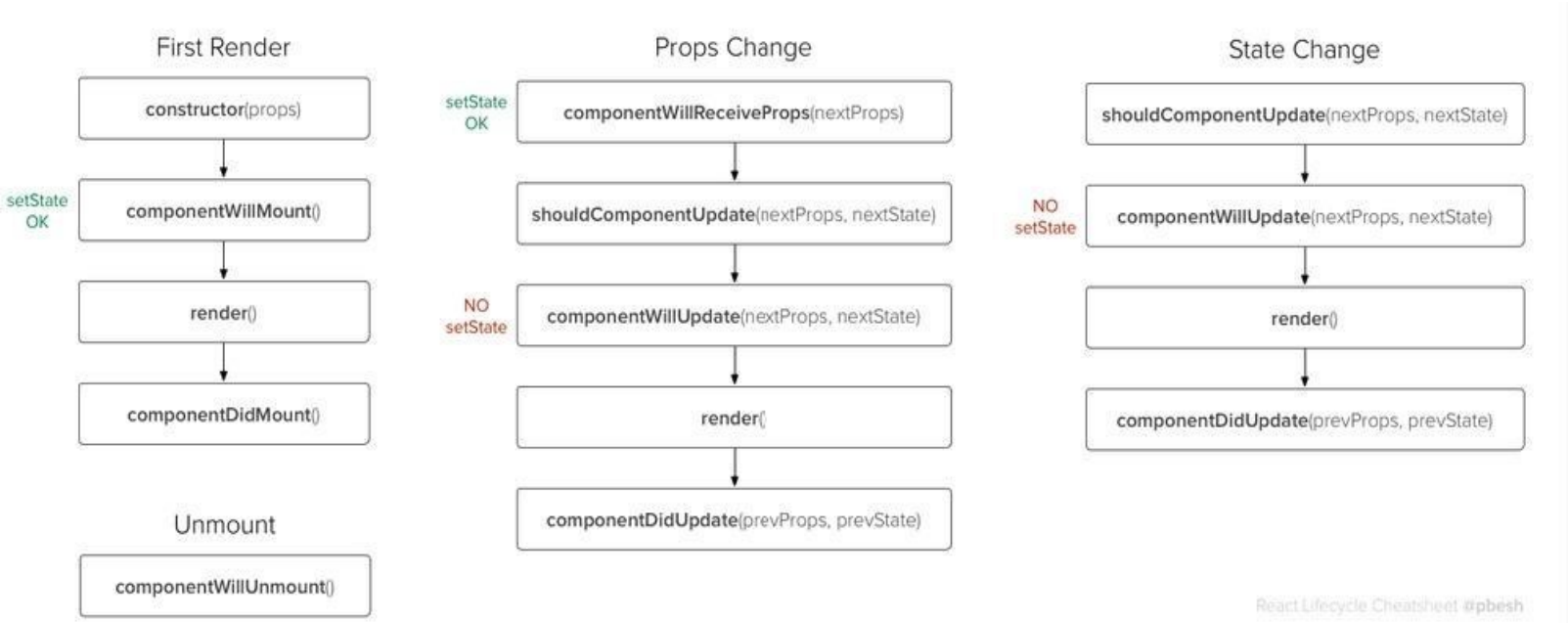
```
class MyComponent extends Component {

    componentDidUpdate() {
        // TODO: add code
    }
}
```

Вызывается каждый раз после обновления компонента (кроме первого рендера).

Вызывать setState можно, но он приведёт к дополнительном рендеру.

OTUS

React lifecycle (что хочется сделать? Где?)



Events

```
// Пример работы с событиями, когда нужно подписаться вне React

class MyComponent extends Component {

  componentDidMount() {
      window.addEventListener('resize', this.handleResize);
  }

  componentWillUnmount() {
      window.removeEventListener('resize', this.handleResize);
  }

  handleResize(e) {
      this.setState({windowWidth: e.width});
  }

  render() {
      return <div>Current window width: {this.state.windowWidth}</div>;
  }
}
```

Events

```
// onClick обработчик
<div onClick={(event) => console.log('!')}>
    Click me!
</div>
```

В случае, если мы хотим работать с событиями компонент в React, то есть намного более удобный способ.

React предоставляет прекрасное API для работы с событиями

React events

- onCopy onCut onPaste
- onCompositionEnd onCompositionStart onCompositionUpdate
- onKeyDown onKeyPress onKeyUp
- onFocus onBlur
- onChange onInput onSubmit

React events

- onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp

React events

- onTouchCancel onTouchEnd onTouchMove onTouchStart
- onScroll
- onWheel
- onLoad onError
- onAnimationStart onAnimationEnd onAnimationIteration
- onTransitionEnd

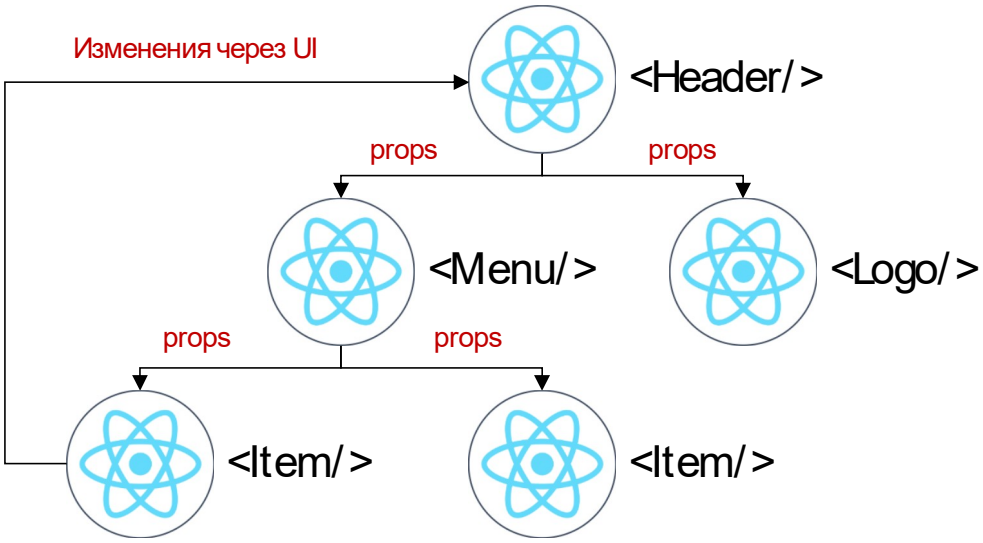
React events

- onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied onEncrypted onEnded onError onLoadedData onLoadedMetadata

SyntheticEvent

- В каждый обработчик приходит не обычное браузерное событие, а обёртка – SyntheticEvent
- Данная обёртка является кросс-браузерной. И если там есть поле, например, keyCode, то оно будет во всех браузерах
- Также есть обычные методы событий: stopPropagation() и preventDefault()
- Если необходимо добраться до нативного события, просто используйте поле nativeEvent

One-way Data-flow



One-way Data-flow

```
class MyComponent extends Component {
    render() {
        return (
            <Button onClick={() => console.log('!')}>
                Click me!
            </Button>
        )
    }
}

class Button extends Component {
    render() {
        return (
            <div onClick={this.props.onClick}>
                {this.props.children}
            </div>
        )
    }
}
```

One-way Data-flow

```
class MyComponent extends Component {
    handleOnClick() {
        console.log('!')
    }
    render() {
        return (
            <Button onClick={this.handleOnClick.bind(this)}>
                Click me!
            </Button>
        )
    }
}

class Button extends Component {
    render() {
        return (
            <div onClick={this.props.onClick}>
                {this.props.children}
            </div>
        )
    }
}
```

Вопросы?

Упражнение

1. create-react-app (или с прошлого занятия)
2. Компонента, показывающая текущее дату/время и пишущая его же в консоль.
3. Данные обновляются раз в секунду

Вопросы?

state and props

Различие между state и props

1. Props-ы (Properties, свойства) поступают в компонент из родительского компонента
2. State инициализируется внутри компонента в конструкторе
3. State – внутреннее состояние компонента
4. Props – начальные данные (обычно)
5. Props могут быть провалидированы, state – нет
6. Большинству компонентов не нужен state, только props

Props vs state

Что должно помещаться в state?

1. Данные по которым может вызываться обновление UI
2. Минимальные данные, неоьходимые для логики

Что не долдно помещаться в state?

1. Дубликаты данных из формы
2. React компоненты
3. Любые вычисляемые данные

OTUS

Props vs state (Упражнение)

Назовите по два компонента, какие их данные можно отнести к props, какие ко state?

Props vs state (Best Practices)

this.state.expanded – в меню/дропдайне

this.state.visible – в Popup

this.state.currentTime – в таймерах

И ВСЁ!

(для остального будет Redux)

Задание state

```
class Button extends Component {

    constructor(props) {
        super();
        this.state = {
            title: props.title,
            active: true
        }
    }
}
```

- Перед тем как изменять state, он задаётся в конструкторе объекта напрямую (state поле компонента).
- До вызова первого render (mount) state нельзя никак менять (да и зачем?)
- Далее state не изменяется напрямую, а только через setState метод

Изменение state

```
onClick() {
  this.setState({
    checked: !this.state.checked
  });

  this.setState(prev => ({
      checked: !prev.checked
  }))
}
```

Каждый вызов setState приводит к запуску цикла обновления компонента и последующим re-render-ом

Это асинхронный метод, и обновления будут доступны в компонент в следующем event-цикле

Props

```
class Label extends Component {
    render() {
        return (
            <span>
                {this.props.text}
            </span>
        )
    }
}

// в parent-е
<Label text={'Hello'}/>
```

Props – immutable параметры, которые приходят из родительского компонента в дочерний. В дочернем доступен через this.props

Единственный правильный способ поменять props – это сказать parent-у (через callback) чтобы он их поменял

Stateless components

```
const Title = (props) => {
    return <span>
        {this.props.text}
    </span>;
};

const Title = ({text}) => <span>{text}</span>;
```

Нет state-а. Просто пишутся. По сути – просто render.

Всё равно будет render, если свойства не изменились

(в будущих версиях React, возможно, будет оптимизировано)

Часто таким образом пишут простые компоненты

OTUS

Компоненты и контейнеры

- Обычно все React компоненты делятся на контейнеры (Containers) и простые компоненты (Dumb Components).
- Контейнеры содержат простые компоненты и другие контейнеры.
- Простые компоненты просто генерируют Virtual DOM по входным данным.
- Такое разделение позволяет:
  - Разделить получение данных и рендеринг этих данных
  - Легко переиспользовать компоненты
  - Проще осуществлять валидацию

Вопросы?

Упражнение

1. create-react-app
2. Компонента, показывающая текущее дату/время и пишущая его же в консоль.
3. Данные обновляются раз в секунду
4. Кнопка, скрывающая/показывающая компоненту времени
5. При скрытии в консоль ничего не должно писаться!
6. **visible** – **state** корневого компонента

Вопросы?

Прочее

defaultProps

```
class Label extends Component {
    render() {
        return <a>{this.props.text}<a>
    }
}

Label.defaultProps = {
    text: 'N/A'
};

// в parent-е
<Label/>
```

defaultProps – свойства по умолчанию. Эти значения подставятся в случае undefined в соответствующем ключе

С помощью них Вы можете безопасно писать компоненты, если даже родитель не передаст Важные свойства.

Обычно используются для массивов или специальных значений
С ними тесты без warning-ов

PropTypes

- Ваше приложение растёт и развивается, то было бы неплохо убедиться, что компоненты работают правильно.
- propTypes - это набор валидаторов, которые позволяют проверить тип, состав полей и просто наличие свойств компонента.
- Работает в development react-e, по сути, выводит только warnigns на консоль браузера.
- Немаловажное достоинство – с помощью propTypes, по сути, документируются props. Подсказки, IntelliJ/WebStorm

One-way Data-flow

```
import PropTypes from "prop-types";

class Label extends Component {}

Label.defaultProps = {
    text: 'N/A'
};

Label.propTypes = {
    text: PropTypes.string
};
```

- Тьма различных валидаторов
- Можно комбинировать валидаторы
- Можно даже проверять форму (наличие полей)

https://www.npmjs.com/package/prop-types

One-way Data-flow

```
render() {
    return <TextInput
        ref={(c) => this._input = c}
    />;
}

componentDidMount() {
    this._input.focus();
}
```

- React поддерживает очень специальные свойства, называемые refs
- С их помощью можно получить доступ к компонентам вне render
- Полезно, если Вам нужно найти какой-то элемент DOM

- Это называется Uncontolled Inputs. Полезно, если Вы прикручиваете React к другому фреймворку.

Упражнение

1. PropTypes и defaultProps для Ваших компонент

Вопросы?

Домашнее задание

На весь блок React:
Приложение для самостоятельной работы в блоке React - веб-приложение погоды. На странице приложения должна быть возможность добавлять города в список избранных. По каждому городу показывается информация о температуре, ветре, другие параметры.

ДЗ сегодня: Создать структуру приложения, создать компоненты контейнеры.

Спасибо за внимание!

Красивых компонент!