React Advanced

Part 2

Рассматриваемые темы

- Render Props
- Context API
- Portal
- Error Boundaries

Render 👀 Props

Render Props — это способ **повторного использования логики** в React и **сокрытие** этой логики от потребителя.

Render Props — это паттерн

```
<Button prop={value}/>
```

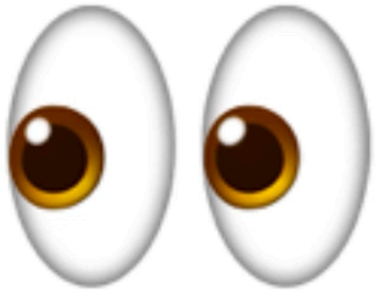
Function, Number, Boolean, String

Функция высшего порядка — функция, которая может возвращать другую функцию

или которая может в **качестве аргумента** принимать **другую функцию.**

```
const Button = ({prop}) => {
    /* smth */
}
```

Передача функции как аргумента — возможность **расширять** компонент не модифицируя его



Higher Order Component

```
const withMouseTracker = Component =>
    class extends React.Component {
        // ->

        render() {
            return (
                <span ref={ref => (this.wrapper = ref)}>
                    <Component
                        deltas={this.state}
                        {...this.props}
                    />
                </span>
            );
        }
    };
```

```
state = {
    deltaX: 0,
    deltaY: 0
};

handleMove = (e) => {
    this.setState({deltaX, deltaY}); // Some logic
};

componentDidMount() {
    document.addEventListener("mousemove", this.handleMove);
}

componentWillUnmount() {
    document.removeEventListener("mousemove", this.handleMove);
}
```

Статическая композиция

```
const ProjectEyes = withMouseTracker(
    Eyes
);
```

```
import {
    withSpace,
    withGalaxy,
    withSolarSystem
} from "@my/universe";
import { Earth } from "./components";

export default withSpace(
    withGalaxy(
        withSolarSystem(Earth)
    )
);
```

Render Props

```
class MouseTracker extends React.Component {
    // setup events and handler
    render() {
        const { render } = this.props;

        return (
            <span ref={ref => (this.wrapper = ref)}>
                {render(this.state)}
            </span>
        );
    }
}
```

Применение

```
import React from "react";

export const MousePosition = deltas => (
    <div class="measures">
        {Object.values(deltas).join(",")}
    </div>
);
```

Применение

```
// <…>
<MouseTracker render={MousePosition} />
<MouseTracker render={Eyes} />
// <...>
```

Render Children Props

```
function Greating({children}) {
    return <div>Hello, {children}</div>
}

function App() {
    return (
        <div className="App">
            <Greating>World</Greating>
        </div>
    );
}
```

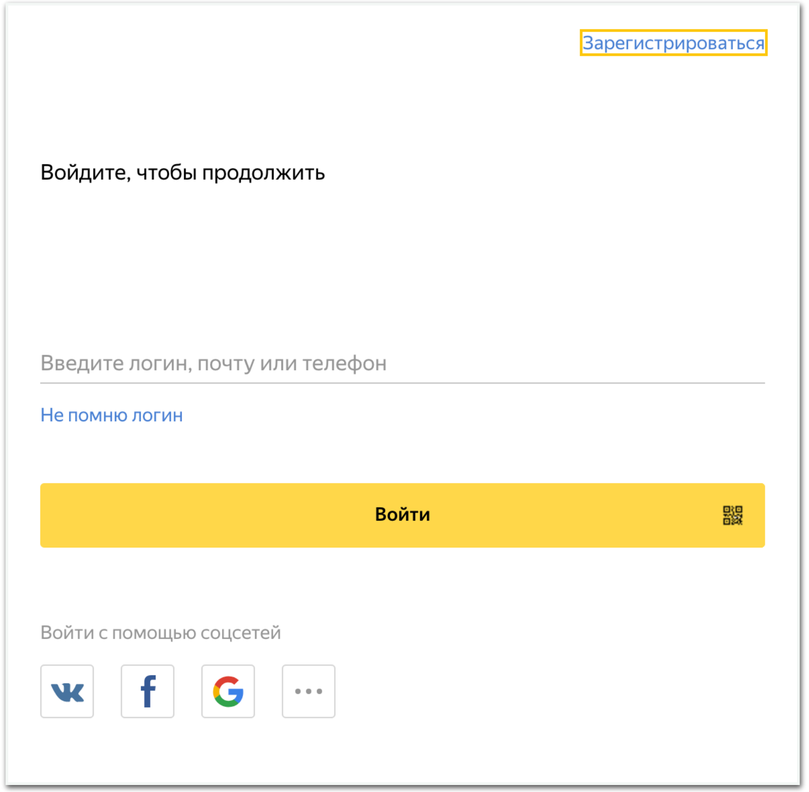
```
function Greating({children}) {
    return <div>Hello, {children}</div>
}

function App() {
    return (
        <div className="App">
            <Greating children='World' />
        </div>
    );
}
```

```
<MouseTracker>
    {(deltas) => <MousePosition deltas={deltas}/>}
</MouseTracker>
```

Еще пример

Войти

Привет, вы не авторизованы Войти



```
class AuthLauncher extends React.Component {
      launchAuthWindow() {
          // Some logic
      }
      render() {
          return this.props.children(
              this.launchAuthWindow.bind(this)
          )
      }
}
```

```
<AuthLauncher>
    {launchAuthWindow => (
        <Button onClick={launchAuthWindow}>Войти</Button>
    )}
</AuthLauncher>
```

```
<AuthLauncher>
    {launchAuthWindow => (
        <div>
            Вы не авторизованы, <a onClick={launchAuthWindow}>Войти</a>
        </div>
    )}
</AuthLauncher>
```

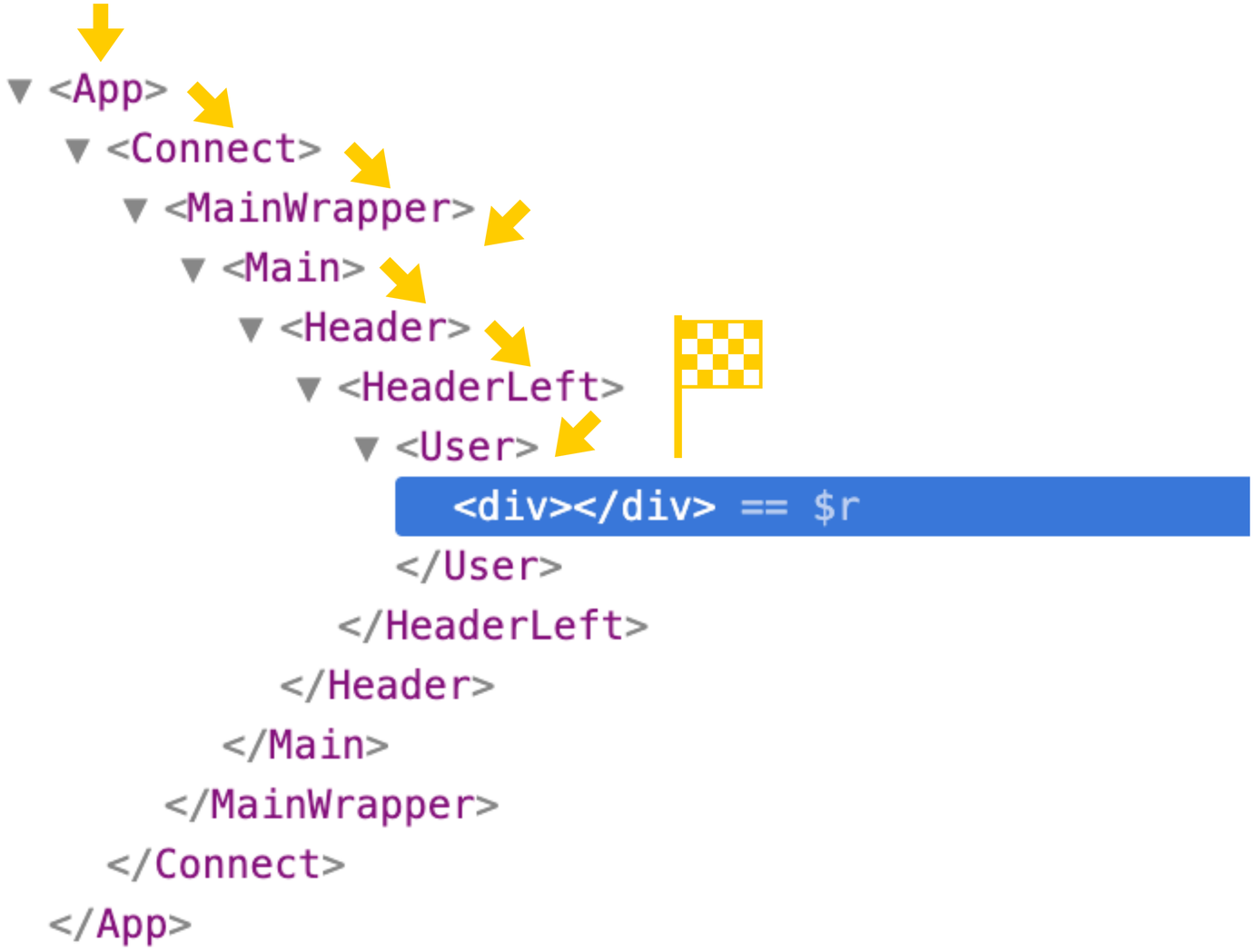
```
<AuthLauncher>
    {launchAuthWindow => (
        <Button onClick={
            () => flow(someAction, anotherAction, launchAuthWindow)
        }>
          Войти
        </Button>
    )}
</AuthLauncher>
```

Render Props это

- Способ повторного использования логики
- Пример динамической композиции

Сильная сторона **JS**X заложено в первых символах этой аббревиатуры

Context API


▼ <App>
▼ <Connect>
▼ <MainWrapper>
▼ <Main>
▼ <Header>
▼ <HeaderLeft>
▼ <User>
 == $r
</User>
</HeaderLeft>
</Header>
</Main>
</MainWrapper>
</Connect>
</App>

Контекст — это сущность, позволяющая передавать пропсы не передавая их через каждую компоненту в дереве.

Хорошие кейсы для Context API

- Темизация приложения
- Пользовательские данные (Profile/User Data)
- Настройки локализации
- Геолокация
- ...

Настройки локализации

```
import React, { createContext } from 'react';

const LanguageContext = createContext({lang: 'en'});

const App = () => (
    <LanguageContext.Provider value={{lang: languageFromServer}}>
        <Connect>
            <MainWrapper>
                <Main>
                    {/* ... */}
                    <LanguageContext.Consumer>
                        {({lang}) => lang === 'ru' ? 'Привет!': 'Hello!'}
                    </LanguageContext.Consumer>
                    {/* ... */}
                </Main>
            </MainWrapper>
        </Connect>
    </LanguageContext.Provider>
);
```

Пользовательские данные

```
import React, { createContext } from 'react';

const Auth = createContext();

const User = ({isAuthenticated}) => (
    isAuthenticated
        ? <h1>Dratuti, known Person!</h1>
        : <button>Log in</button>
)

const App = () => (
    <Auth.Provider value={{isAuthenticated: true}}>
        <MainWrapper>
            <Main>
                {/* ... */}
                <Auth.Consumer>
                    {User}
                </Auth.Consumer>
                {/* ... */}
            </Main>
        </MainWrapper>
    </Auth.Provider>
);
```

```
const App = () => (
    <Auth.Provider value={{isAuthenticated: true}}>
        <Connect>
            {/* ... */}
            <Auth.Consumer>
                {User}
            </Auth.Consumer>
            {/* ... */}
            <Auth.Consumer>
                {CreatePostForm}
            </Auth.Consumer>
        </Connect>
    </Auth.Provider>
);
```

Контексты могут быть вложенными и чем **ближе** к потребителю контекст, тем он **приоритетнее**

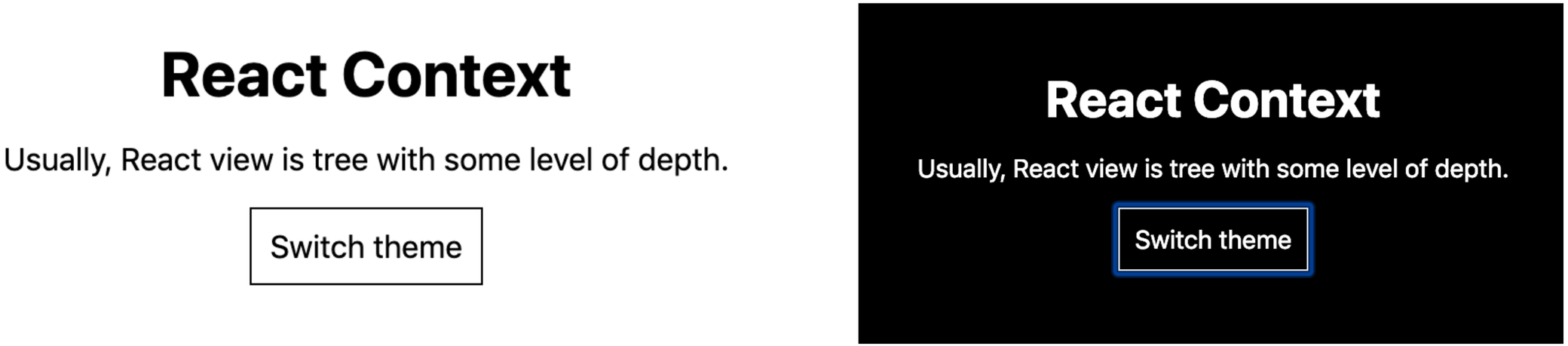
```
const ColorContext = createContext();

const User = (style) => <h1 style={style}>User</h1>

function App() {
    return (
        <ColorContext.Provider value={{color: 'green', backgroundColor: 'red'}}>
            <ColorContext.Provider value={{color: 'red'}}>
                <ColorContext.Consumer>
                    {User}
                </ColorContext.Consumer>
            </ColorContext.Provider>
        </ColorContext.Provider>
    );
}
```

Контекстов может быть **несколько**.

Потребители в таком случае могут быть **вложенными** друг в друга

```
return (
    <ThemeContext.Provider value={theme}>
        <UserContext.Provider value={signedInUser}>
            <Content />
        </UserContext.Provider>
    </ThemeContext.Provider>
);
```

```
function Content() {
    return (
        <ThemeContext.Consumer>
            {theme => (
                <UserContext.Consumer>
                    {user => (
                        <ProfilePage user={user} theme={theme} />
                    )}
                </UserContext.Consumer>
            )}
        </ThemeContext.Consumer>
    );
}
```


React Context
Usually, React view is tree with some level of depth.
Switch theme
React Context
Usually, React view is tree with some level of depth.
Switch theme

```
const ThemeContext = createContext({
    value: "light",
    changeTheme: () => {}
});
```

```
function* getThemeGenerator() {
    let theme = "dark";
    while (true) {
        yield (theme = theme === "light" ? "dark" : "light");
    }
}
```

```
class Container extends React.Component {
    theme = getThemeGenerator();
    getTheme = () => this.theme.next().value;

    changeTheme = () => {
        this.setState(state => ({
            theme: {
                ...state.theme,
                value: this.getTheme()
            }
        }));
    };

    state = {
        theme: {
            value: this.getTheme(),
            changeTheme: this.changeTheme
        }
    };
}
```

```
class Container extends React.Component {
    // . . .
    render() {
        return (
            <ThemeContext.Provider value={this.state.theme}>
                <App>
                    <h1 className="Heading">React Context</h1>
                    <p>Usually, React view is tree with some level of depth.</p>
                    <div>
                        <Button>Switch theme</Button>
                    </div>
                </App>
            </ThemeContext.Provider>
        );
    }
}
```

```
class App extends React.Component {
    static contextType = ThemeContext;

    getStyle(theme) {
        const isLight = this.context.value === "light";

        return {
            color: isLight ? "black" : "white",
            backgroundColor: isLight ? "white" : "black"
        };
    }
    render() {
        return (
            <div style={this.getStyle()} className="App">
                {this.props.children}
            </div>
        );
    }
}
```

```
class Button extends React.Component {
    static contextType = ThemeContext;

    getStyle() {
        const isLight = this.context.value === "light";

        return {
            borderColor: isLight ? "black" : "white",
            color: isLight ? "black" : "white",
            backgroundColor: isLight ? "white" : "black"
        };
    }

    render() {
        return (
            <button
                onClick={() => this.context.changeTheme()}
                style={this.getStyle()}
                className="Button"
                {...this.props}
            >
                {this.props.children}
            </button>
        );
    }
}
```

Контекстом не следует пренебрегать. ~95% пропсов, которые вы передаете следует передавать явно.

Альтернативы контексту

- Компоненты-контейнеры (из Redux)
- Хорошая композиция

Про хорошую композицию

```
<Page user={user} avatarSize={avatarSize} />
// ... Который рендерит ...
<PageLayout user={user} avatarSize={avatarSize} />
// ... Который рендерит ...
<NavigationBar user={user} avatarSize={avatarSize} />
// ... Который рендерит ...
<Link href={user.permalink}>
    <Avatar user={user} size={avatarSize} />
</Link>
```

```
function Page(props) {
    const user = props.user;
    const userLink = (
        <Link href={user.permalink}>
            <Avatar user={user} size={props.avatarSize} />
        </Link>
    );

    return <PageLayout userLink={userLink} />;
}
```

И теперь

```
<Page user={user} avatarSize={avatarSize} />
// ... который рендерит ...
<PageLayout userLink={...} />
// ... который рендерит ...
<NavigationBar userLink={...} />
// ... который рендерит ...
{props.userLink}
```

Portals

Портал — способ рендера компонента вне общего дерева рендеринга.

```
render() {
    // В нормальном режиме React монтирует дочерние узлы
    // в ближайшего верхнего родителя в DOM
    return (
        <main className="main">
            <div className="main__content">
                <div className="user"></div>
            </div>
        </main>
    );
}
```

```
import ReactDOM from 'react-dom';
```

```
<html>
  <body>
    <div id="app-root"></div>
    <div id="modal-root"></div>
  </body>
</html>
```

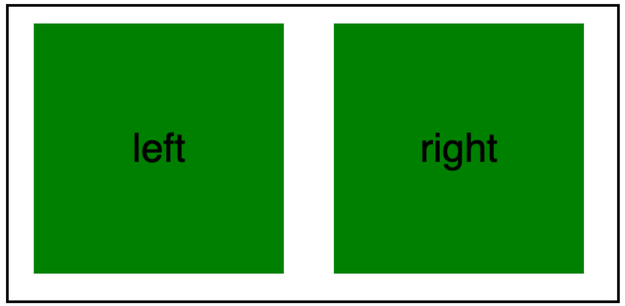
```
class App extends React.Component {
    render() {
        return (
            <div>
                <Modal>
                    <Child />
                </Modal>
            </div>
        );
    }
}
```

```
class Modal extends Component {
    render() {
        const visible = this.props.isVisible ? "yes" : "no";

        return ReactDOM.createPortal(
            <div className={modal({ visible })}>
                <div className={modal("content")}>{this.props.children}</div>
            </div>,
            document.getElementById("modal-root")
        );
    }
}
```

События в портале

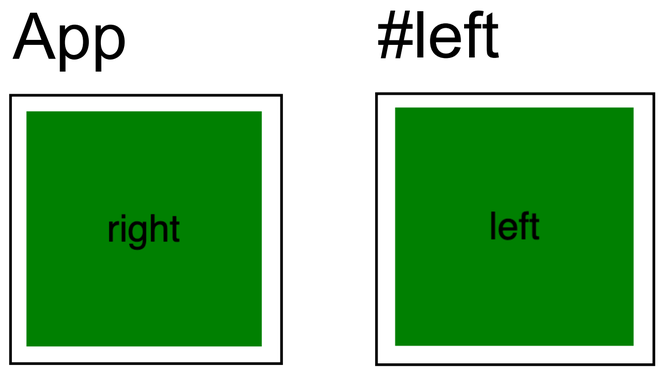
```jsx
function App() {
    return (
        <div className="App">
            <div className="left" onClick={() => console.log("left")}>
                <div className="left__inner" onClick={() => console.log("left__inner")}>
                    left
                </div>
            </div>
            <div className="right" onClick={() => console.log("right")}>
                <div
                    className="right__inner"
                    onClick={() => console.log("right__inner")}
                >
                    right
                </div>
            </div>
        </div>
    );
}
```

App




```
right__inner
right
app
left__inner
left
app
```

```jsx
function App() {
    return (
        <div className="App" onClick={() => console.log("app")}>
            {ReactDOM.createPortal(
                <div className="left" onClick={() => console.log("left")}>
                    <div
                        className="left__inner"
                        onClick={() => console.log("left__inner")}
                    >
                        left
                    </div>
                </div>,
                document.getElementById("left")
            )}
            <div className="right" onClick={() => console.log("right")}>
                <div
                    className="right__inner"
                    onClick={() => console.log("right__inner")}
                >
                    right
                </div>
            </div>
        </div>
    );
}
```




```
right__inner
right
app
left__inner
left
app
```

Error Boundaries

Ранее ошибки JavaScript внутри компонентов портили внутреннее состояние. Для того, чтобы обрабатывать ошибки придумали **компоненты-предохранители** (**Error Boundaries**)

```
class App extends React.Component {
    componentDidMount() {
        return notExisten;
    }

    render() {
        return <div>{`Hello, ${value}`}</div>;
    }
}
```

```
class ErrorBoundary extends React.Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false };
    }

    static getDerivedStateFromError(error) {
        // Update state so the next render will show the fallback UI.
        return { hasError: true };
    }

    componentDidCatch(error, info) {
        // You can also log the error to an error reporting service
        console.log(error, info);
    }

    render() {
        if (this.state.hasError) {
            // You can render any custom fallback UI
            return <h1>Something went wrong.</h1>;
        }

        return this.props.children;
    }
}
```

```
ReactDOM.render(
    <ErrorBoundary>
        <App />
    </ErrorBoundary>
, document.getElementById("root"));
```

Типы необрабатываемых ошибок

- Обработчики событий (onChange, onMouseMove etc.)
- Асинхронный код (setTimeout, setInterval, etc.)
- Рендеринг на сервере
- Ошибки на самом компоненте-*предохранителе*

Где размещать компоненты-предохранители

- На самом верхнем уровне — общим для всех
- Вокруг *подозрительного* компонента

???

Спасибо

Александр Шоронов

Разработчик интерфейсов

@underoot