Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 15

Children. Что такое context. HOC

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Children
2. Context
3. HOC

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

В React компонент может иметь одного, многих или ни одного дочернего элемента

```
<Profile>
  <ProfileImage src="/asset/profile-img.png" />
  <ProfileDetails name="Antonello" surname="Zanini" />
</Profile>
```

Компонент Profile имеет двух дочерних элементов: ProfileImage и ProfileDetails, в то время как эти два не имеют дочерних элементов

В выражениях JSX, которые содержат как открывающий, так и закрывающий теги, содержимое между этими тегами передается как специальное свойство: props.children – Документация React

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

props.children - это специальная объект, автоматически передаваемая каждому компоненту, которая может использоваться для рендеринга содержимого, включенного между открывающим и закрывающим тегами при вызове компонента. Такие компоненты обозначены в официальной документации как коробки

В React's JSX компонент с дочерними элементами всегда идентифицируется открывающим и закрывающим тегами

Каждый дочерний элемент должен быть помещен между этими двумя тегами.

Когда у компонента нет дочерних элементов, можно вызвать его с помощью <MyComponent></MyComponent> или <MyComponent/>, но последний синтаксис обычно предпочтительнее.

Назначение самозакрывающихся тегов - сделать код короче и удобнее для чтения.

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

props.children в действиях

Создадим ImageSlider компонент. Наша цель - вызвать компонент следующим образом:

```
<ImageSlider>
  <img src="/assets/img-1.pg" />
  <img src="/assets/img-2.pg" />
  <img src="/assets/img-3.pg" />
</ImageSlider>
```

Получим доступ к содержимому:

```
export default function ImageSlider(props) {
    return (
      <div className="img-slider">
        {props.children}
      </div>
    );
}
```

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

props.children в действиях

Создадим ImageSlider компонент. Наша цель - вызвать компонент следующим образом:

```
<ImageSlider>
  <img src="/assets/img-1.pg" />
  <img src="/assets/img-2.pg" />
  <img src="/assets/img-3.pg" />
</ImageSlider>
```

В `props.children` можем вкладывать контент в компонент, как вкладываем обычные HTML-элементы

Получим доступ к содержимому:

```
export default function ImageSlider(props) {
    return (
      <div className="img-slider">
        {props.children}
      </div>
    );
}
```

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

props.children позволяет нам составлять компоненты и, как следствие, наш интерфейс; использование реальной мощи компонентной модели React.

Почему props.children «непрозрачная структура данных»?

props.children может состоять из одного, многих или ни одного дочерних элементов, это означает, что его значение может быть одним дочерним узлом, массивом дочерних узлов или undefined соответственно.

Благодаря React.Children API легко справиться с props.children, не принимая во внимание каждый из его возможных типов.

Сделано это в фоновом режиме.

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Допустим, мы хотим добавить специальный класс CSS img-special-class к каждому из потомков компонента ImageSlider. Это можно сделать следующим образом:

```
export default function ImageSlider(props) {
  const { children } = props

  return (
    <div className="img-slider">
      {
        React.Children.map(children, (child) =>
          React.cloneElement(child, {
              className: `${child.props.className} img-special-class`
            })
        )
      }
    </div>
  );
}
```

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

React.Children.map позволяет перебирать props.children и преобразовывать каждый элемент в соответствии с функцией, переданной в качестве второго параметра

Для это используется React.cloneElement.
Необъодимо изменить значение свойства className, но свойства неизменяемы в React, поэтому пришлось клонировать каждый дочерний элемент

Children

Функция как дочерний элемент

Это конечно не совсем пригодный пример, но он показывает саму идею.

Можем передать любое JavaScript выражение, как дочерний элемент

Компонент, который выполняет функцию, которая была передана как дочерний элемент:

```
class Executioner extends React.Component {
  render() {

    return this.props.children()
  }
}
```

Children

Вызов этого компонента

```
<Executioner>
  {() => <h1>Hello World!</h1>}
</Executioner>
```

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Управление children

Подсчитать количество потомков

```
class ChildrenCounter extends React.Component {
  render() {
    return <p>React.Children.count(this.props.children)</p>
  }
}
```

Children

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Конвертируем children в массив

Можно конвертировать children в массив при помощи метода React.Children.toArray

```
class Sort extends React.Component {
  render() {
    const children = React.Children.toArray(this.props.children)
    return <p>{children.sort().join(' ')}</p>
  }
}
<Sort>
  // Тут мы используем контейнеры выражений, чтобы убедиться в том, что наши
строки
  // переданы как три отдельных потомка, а не как одна строка
  {'bananas'}{'oranges'}{'apples'}
</Sort>
```

Обратите внимание, что возвращаемый React.Children.toArray массив не содержит потомков с типом функция, а только React.Element или строки.

Children

Использовать React.Children.only внутри рендера

```
class Executioner extends React.Component {
  render() {
    return React.Children.only(this.props.children)()
  }
}
```

В этом случае отрендерится только один единственный потомок в this.props.children.

Если их будет больше, чем один, то нам выкинет ошибку, следовательно, работа приложения будет приостановлена

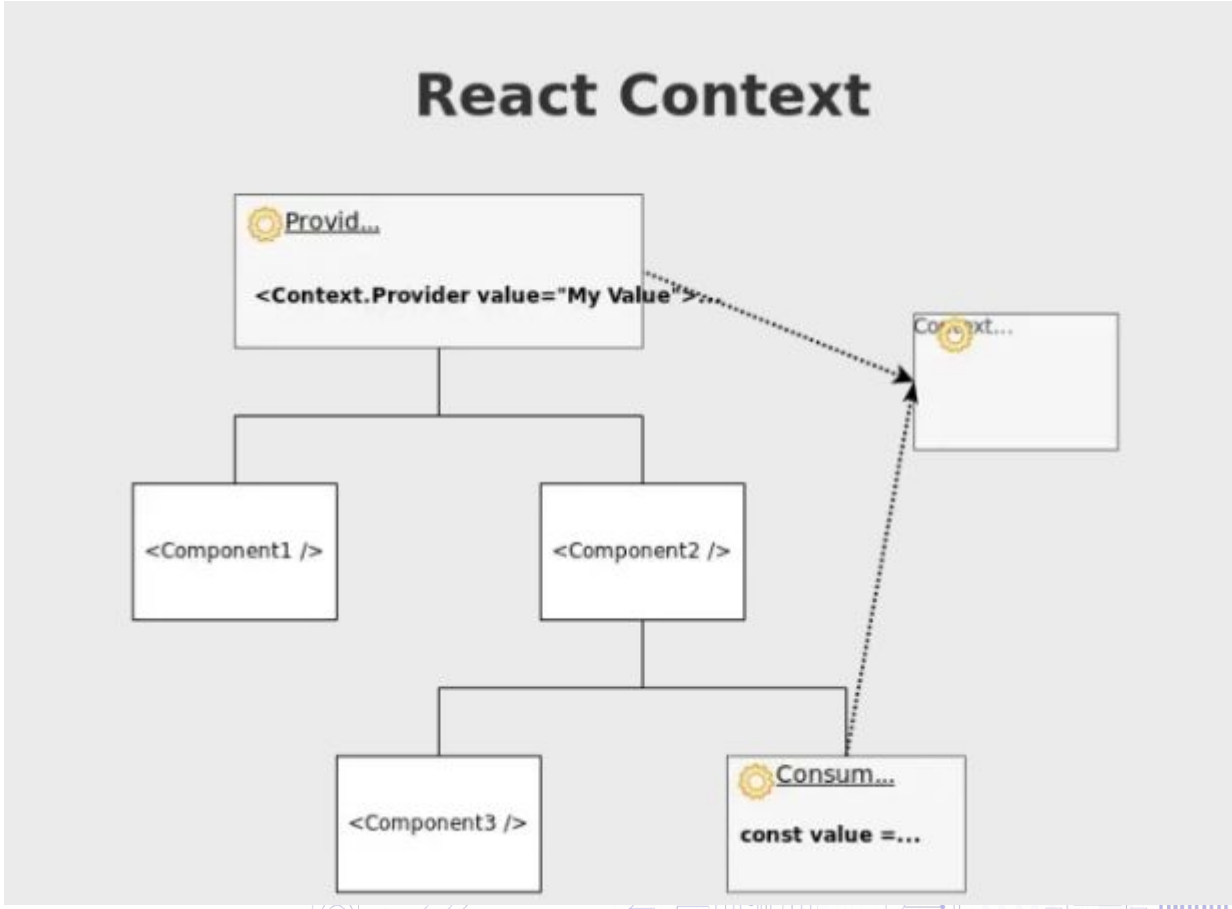
Context

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Контекст React предоставляет данные компонентам независимо от того, насколько глубоко они находятся в дереве компонентов

Для использования контекста в React требуется 3 простых шага:

- создание контекста(А)
- предоставление контекста(Б)
- использование контекста(В)



Context

A.Создадим контекст

Встроенная функция createContext (по умолчанию) создает экземпляр контекста

Функция принимает один необязательный аргумент: 'Любое значение’

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

const Context = createContext('Любое значение')
```

Context

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Б. Предоставление контекста

Компонент Context.Provider — доступный в экземпляре контекста, используется для предоставления контекста его дочерним компонентам, независимо от их глубины.

```
const App = () => {
    const value = 'Что-то положим в контекст'

    return (
        <Context.Provider value={value}>
            компоненты
        </Context.Provider>
    )
}
```

Все компоненты, которые позже захотят использовать контекст, должны быть заключены в компонент провайдера.

Чтобы установить значение контекста, используйте свойство value, доступное в <Context.Provider value = {value} />

Context

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

В. Использование контекста

Использование контекста может быть выполнено двумя способами. Первый способ — это использовать хук useContext (Context)

Хук возвращает значение контекста: value = useContext (Context)

```
const Component = () => {
    const value = useContext(Context)

    return (
        <div>
            {value}
        </div>
    )
}
```

Хук также гарантирует повторный рендеринг компонента при изменении значения контекста

Context

B. Использование контекста

Второй способ — использовать функцию рендеринга, предоставленную в качестве дочернего для специального компонента Context.Consumer, доступного в экземпляре контекста:

```
const Component = () => {
    return (
        <Context.Consumer>
            {value => <div>{value}</div>}
        </Context.Consumer>
    )
}
```

Context

Можно создать столько дочерних компонентов, сколько хотите для одного контекста.

Если значение контекста изменяется (путем изменения свойства value провайдера (), то все дочерние компоненты немедленно уведомляются и повторно обрабатываются.

Если дочерний компонент не заключен внутри провайдера, но все же пытается получить доступ к значению контекста (используя useContext (Context) или) , то значение контекста будет аргументом значения по умолчанию, предоставленным createContext (defaultValue) функция, создавшая контекст.

Context

Когда применять

Основная идея использования контекста — предоставить компонентам доступ к некоторым глобальным данным и повторный рендеринг при изменении этих глобальных данных.

Контекст решает проблему сверления props: когда нужно передать реквизиты от родителей детям

Можете запихнуть внутрь контекста:

- Глобальное состояние приложения
- Тема приложения
- Конфигурация приложения
- Аутентификация пользователя
- Пользовательские настройки
- Предпочтительный язык
- Набор услуг

Context

Не забываем про:

- Во-первых, интеграция контекста добавляет сложности. Создание контекста, обертывание всего в провайдере, использование useContext () в каждом дочернем компоненте — это увеличивает сложность.

- Во-вторых, добавление контекста затрудняет модульное тестирование компонентов. Во время модульного тестирования придется заключить потребительские компоненты в поставщик контекста. Включая компоненты, на которые косвенно влияет контекст — родителей дочерних компонентов контекста!

HOC

Компонент высшего порядка (Higher-Order Component, HOC) — это один из продвинутых способов для повторного использования логики.

HOC не являются частью API React, но часто применяются из-за композиционной природы компонентов.

Компонент высшего порядка — это функция, которая принимает компонент и возвращает новый компонент

```
const EnhancedComponent = higherOrderComponent(
  WrappedComponent
)
```

Обычный компонент преобразует пропсы в UI, то компонент высшего порядка преобразует компонент в другой компонент.

HOC часто встречаются в сторонних библиотеках, например connect в Redux и createFragmentContainer в Relay.

HOC

Функции, которые берут компонент как аргумент и возвращают уже «переработанный» компонент.

HOC будет всегда иметь вид как тут:

```
import React from 'react';

const higherOrderComponent = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return <WrappedComponent />;
    }
  }

  return HOC;
};
```

HOC


```
import React from 'react';

const withSecretToLife = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return (
        <WrappedComponent
          {...this.props}
          secretToLife={42}
        />
      );
    }
  }

  return HOC;
};

export default withSecretToLife;
```

HOC

Стоит обратить внимание, что используем оператор расширения на пропсах переданных компоненту.

Так мы точно будем уверены в том, что любой пропс переданный обернутому компоненту будет доступен через this.props, так же, как и если бы вызывали этот компонент не через компонент высшего порядка

```
import React from 'react';
import withSecretToLife from 'components/withSecretToLife';

const DisplayTheSecret = props => (
  <div>
    The secret to life is {props.secretToLife}.
  </div>
);

const WrappedComponent = withSecretToLife(DisplayTheSecret);

export default WrappedComponent;
```

HOC

Стоит обратить внимание, что используем оператор расширения на пропсах переданных компоненту.

Так мы точно будем уверены в том, что любой пропс переданный обернутому компоненту будет доступен через this.props, так же, как и если бы вызывали этот компонент не через компонент высшего порядка

```
import React from 'react';
import withSecretToLife from 'components/withSecretToLife';

const DisplayTheSecret = props => (
  <div>
    The secret to life is {props.secretToLife}.
  </div>
);

const WrappedComponent = withSecretToLife(DisplayTheSecret);

export default WrappedComponent;
```

Компонент WrappedComponent, который по сути является прокачанной версией <DisplayTheSecret/>, позволит получить доступ к SecretToLife как к пропсу.

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание