this.props.children

this.props.children — це спеціальна властивість, що містить дочірні елементи, що знаходяться між тегами в JSX виразі, а не в самому тегові.

|  |  |
| --- | --- |
|  |  |
| Передача дочірніх компонентів (між тегами батьківського компонента)  <компонент >  опис дочірніх складових (**довільного типу**)  </компонент > | <MyComponent prop1 ={значення1} prop2 ={значення2} >  <a href=’…’ >  <p> Some Text </p>  </MyComponent> |
| У батьківському компоненті усі дочірні складові (довілього типу) знаходяться у властивості  this.props.children  **Виведення дочірніх без змін** | class MyComponent extends Component{  . . . .  render(){  const {prop1, prop2} = this.props  return (  { this.props.children } 🡨 Виведення дочірніх компонентів елементів  )  }  } |
| **Перебір дочірніх із зміною** (необхідно робити клони компонентів, оскільки вихідні компоненти змінювати заборонено)  Перебір дочірніх (створюємо нові елем.)  **React.Children.map**(  this.props.children,  child **=>** { return новий елемент }  -----------------------------------------------------  Створення копій  **React.cloneElement**(  child,  { опис нових властивостей}  ) | **export** **default** **class** MovieBrowser **extends** React.Component {  render() {  **const** currentPlayingTitle **=** 'Mad Max: Fury Road';  **const** childrenWithExtraProp **=**  **React.Children.map**(  **this**.props.children,  child **=>** {  return  **React.cloneElement**(  child,  {isPlaying: child.props.title **===** currentPlayingTitle}  );  });  return (  **<**div className**=**"movie-browser"**>**  {childrenWithExtraProp}  **<**/div>  );  }  } |
| **Підрахунок кількості дочірніх елементів**  **React.Children.count**(**this**.props.children) | **export** **default** **class** SomeComponent **extends** React.Component {  render() {  **const** childrenWithWrapperDiv **=** React.Children.map(**this**.props.children, child **=>** {  **return** (  **<**div className**=**"some-component-special-class"**>**{child}**<**/div>  );  });  **return** (  **<**div className**=**"some-component"**>**  **<**p**>**This component has  {**React.Children.count**(**this**.props.children)} children.  **<**/p>  {childrenWithWrapperDiv}  **<**/div>  );  }  } |

<https://learn.co/lessons/react-this-props-children>

**React This Props Children**

**Overview**

We'll cover what this.props.children means in the context of a React component.

**Objectives**

1. Use this.props.children to render content in our component
2. Explain how to use the React.Children utilities
3. Use and iterate over child components

**Children in a component**

In React, a component can have one, many or no children. Consider the following code:

**<**VideoPlayer**>**

**<**VideoHeader**>**

**<**h1 className**=**"video-title"**>**The Simpsons**<**/h1>

**<**/VideoHeader>

**<**VideoControls **/>**

**<**/VideoPlayer>

In this example, the VideoPlayer has two children: VideoHeader and VideoControls. VideoHeader, in turn, has one child: the h1with the title content. VideoControls, on the other hand, has no children.

Why is this important? As you can see above, we can use children to compose our interface. For a more concrete example, let's say we're creating a <Panel> component that allows us to add content to it. Using a panel might look a little like this:

**<**Panel title**=**"Browse for movies"**>**

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**/Panel>

As you can see, we're adding content *inside* of the <Panel> tags. Now, how do we render that content in our component? We access it through **this.props.children** — a special prop that is passed to components automatically.

**export** **default** **class** Panel **extends** React.Component {

render() {

**return** (

**<**div className**=**"panel"**>**

**<**div className**=**"panel-header"**>**{**this**.props.title}**<**/div>

**<**div className**=**"panel-body"**>**{**this**.props.children}**<**/div>

**<**/div>

);

}

}

If something like this.props.children didn't exist, we'd have to pass in all of our content through a prop, which would be very unwieldy and look really ugly:

**<**Panel

title**=**"Browse for movies"

body**=**{

**<**div**>**

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**div**>**Movie stuff...**<**/div>

**<**/div>

}

/>

*And* we'd have to wrap it in an enclosing div! Thankfully, we can just nest it inside of the component like we did above, much like we nest regular HTML elements.

**React.Children**

Since this.props.children can have one element, multiple elements, or none at all, its value is respectively a single child node, an array of child nodes or undefined. Sometimes, we want to transform our children before rendering them — for example, to add additional props to every child. If we wanted to do that, we'd have to take the possible types of this.props.children into account. For example, if there is only one child, we can't map it.

Luckily, React provides us with a clean API to handle looping of children. If there is only one child (or none at all), it won't throw a fuss — it'll handle things for us nicely in the background.

Let's say we have a list of Movie components that are nested inside of a MovieBrowser component:

**<**MovieBrowser**>**

**<**Movie title**=**"Mad Max: Fury Road" **/>**

**<**Movie title**=**"Harry Potter & The Goblet Of Fire" **/>**

**<**/MovieBrowser>

Now, let's assume for some reason that we need to pass down an extra prop to our children — the props would like to know if they are being played or not. Our MovieBrowser component would look something like this, before we added the prop:

**export** **default** **class** MovieBrowser **extends** React.Component {

render() {

**const** currentPlayingTitle **=** 'Mad Max: Fury Road';

**return** (

**<**div className**=**"movie-browser"**>**

{**this**.props.children}

**<**/div>

);

}

}

Now let's add in our isPlaying prop to the children of MovieBrowser:

**export** **default** **class** MovieBrowser **extends** React.Component {

render() {

**const** currentPlayingTitle **=** 'Mad Max: Fury Road';

**const** childrenWithExtraProp **=** **React.Children.map**(**this**.props.children, child **=>** {

**return** React.cloneElement(child, {

isPlaying: child.props.title **===** currentPlayingTitle

});

});

**return** (

**<**div className**=**"movie-browser"**>**

{childrenWithExtraProp}

**<**/div>

);

}

}

React.Children.map has two parameters: the first one is the children themselves, and the second one is a function that transforms the value of the child. In this case, we're adding an extra prop. We do that using React.cloneElement. As the first argument we pass in the child component itself, and as the second argument, we pass in any additional props. Those additional props get merged with the child's existing props, overwriting any props with the same key.

**More iteration**

As another example, let's say we want to wrap our components in an extra div with a special class. We also want to display the total amount of children.

**export** **default** **class** SomeComponent **extends** React.Component {

render() {

**const** childrenWithWrapperDiv **=** React.Children.map(**this**.props.children, child **=>** {

**return** (

**<**div className**=**"some-component-special-class"**>**{child}**<**/div>

);

});

**return** (

**<**div className**=**"some-component"**>**

**<**p**>**This component has {**React.Children.count**(**this**.props.children)} children.**<**/p>

{childrenWithWrapperDiv}

**<**/div>

);

}

}

**Resources**

* [Explanation on Children](https://facebook.github.io/react/docs/multiple-components.html#children)
* [React.Children API](https://facebook.github.io/react/docs/top-level-api.html#react.children)