React: componenti e DOM

React è stato creato per gestire le interfacce utente suddividendole in diversi componenti riutilizzabili. Ogni componente è un elemento personalizzato la cui rappresentazione DOM viene convertita in una struttura DOM più canonica.

Poiché il DOM si basa su una gerarchia ad albero e una relazione tra elementi genitori ed elementi figli, React aderisce alle specifiche DOM creando una gerarchia di componenti, ma con una regola ferrea quando si tratta di JSX: se ci sono elementi fratelli, questi **devono** avere sempre un elemento genitore. Non si possono ad esempio definire due elementi come i seguenti:

```
return (
        <h1>Items</h1>
        <ul className="items">
            {items.map(({ name, price }, index) => (
                 <li key={index}>
                   { name } / { price }
                 </li>
            ))}
        </ul>
);
```

`h1` e `ul` devono essere racchiusi all'interno di un elemento genitore. Si può utilizzare un elemento del DOM o un contenitore generico (`<></>`), in questo modo:

```
return (
    <>
        <h1>Items</h1>
        <ul className="items">
            {items.map(({ name, price }, index) => (
                 <li key={index}>
                   { name } / { price }
                 </li>
            ))}
        </ul>
    </>
);
```

Finché si comprende questa relazione genitore-figlio, non si avranno problemi con la creazione dei componenti React. Ma come funzionano i componenti? In breve, un componente è solo una funzione che restituisce un'espressione. Se si denomina `Test` la propria funzione, allora si potrà usare un componente chiamato `<Test />` in altre parti del codice proprio come qualsiasi altro elemento del DOM (`<p>`, `<div>`, ecc.).

Un componente può o non può avere figli. Se usiamo `<Test />` senza discendenti, dobbiamo seguire la regola adottata da JSX e utilizzare la sintassi XML `/>` degli elementi vuoti. Questa regola si applica anche ai normali elementi DOM vuoti, come `br`, `input`, `hr` e `img`.

Come detto in precedenza, stiamo usando una funzione. Internamente, React passa alla nostra funzione un argomento comunemente noto come `props`, che è un oggetto contenente sia proprietà predefinite che proprietà personalizzate. Per esempio, se vogliamo creare un componente contenitore per il nostro componente `<Test/>`, possiamo utilizzare la proprietà predefinita `children` che consente al nostro contenitore di ospitare i suoi discendenti.

```
export default function Wrapper(props) {
    return (
        <div className="wrapper">
            { props.children }
        </div>
    );
}
```

Ora possiamo usare il nostro contenitore con il componente `<Test />`:

```
import Wrapper from './components/Wrapper';
import Test from './components/Test';

export default function App() {
    return (
        <Wrapper>
            <Test />
        </Wrapper>
    );
}
```

Abbiamo menzionato prima l'oggetto `props`. Questo oggetto è fondamentale per far comunicare tra loro i componenti. In React ogni componente è isolato e ha il suo **state**. La funzione `useState()` è chiamata **hook** e serve a definire una variabile e la sua funzione setter all'interno di un componente.

```
const [items, setItems] = useState([]);
```

Con questa dichiarazione, definiamo l'array `items` e la sua funzione setter `setItems()` tramite l'array destructuring. L'array passato come argomento a `useState()` definisce il valore iniziale (o state) di `items`, cioè un array vuoto.

Si tenga presente che senza la funzione setter `setItems()` non possiamo impostare o sovrascrivere direttamente il valore iniziale di `items`. Infatti, vediamo come possiamo utilizzare questa funzione setter all'interno dell'hook `useEffect()` di React:

```
useEffect(() => {
        axios.get('/api/items').then(resp => {
            setItems(resp.data);
            // Ora items[] è stato impostato
        });
}, []);
```

Come afferma la documentazione di React a proposito dell'hook `useEffect()`:

> What does useEffect do? By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed (we'll refer to it as our

> “effect”), and call it later after performing the DOM updates. In this effect, we set the document title, but we could also perform data fetching or call some other imperative API.

Qui la cosa fondamentale da capire è che senza la chiamata a `setItems()` dopo la nostra richiesta API, `items` rimarrebbe un array vuoto.

Quindi, come possono i componenti comunicare tra loro se i loro state sono isolati? La risposta è semplice: tramite le proprietà personalizzate che definiamo in un componente genitore. Diciamo che vogliamo passare un valore dallo state di un componente genitore al componente `<Test />`. Per prima cosa, dobbiamo definire il nostro valore nel componente genitore.

```
import { useState, useEffect } from 'react';
import Wrapper from './components/Wrapper';
import Test from './components/Test';

export default function App() {
    const [title, setTitle] = useState('');
    // useEffect() è qui usato a scopo dimostrativo
    useEffect(() => {
         setTitle('Risultati')
    }, []);
    return (
        <Wrapper>
            <Test />
        </Wrapper>
    );
}
```

Ora possiamo passare il valore della variabile `title` come attributo personalizzato al componente `<Test/>`. Questa modifica renderà la proprietà `title` disponibile nell'oggetto `props` del componente `Test`.

```
import { useState, useEffect } from 'react';
import Wrapper from './components/Wrapper';
import Test from './components/Test';

export default function App() {
    const [title, setTitle] = useState('');
    // useEffect() è qui usato a scopo dimostrativo
    useEffect(() => {
         setTitle('Risultati')
    }, []);
    return (
        <Wrapper>
            <Test title={title} />
        </Wrapper>
    );
}
```

Infine, nella funzione `Test` possiamo usare questa proprietà come segue:

```
import { useState, useEffect } from 'react';
import axios from 'axios';

// Usiamo l'object destructuring su props per ottenere title
```

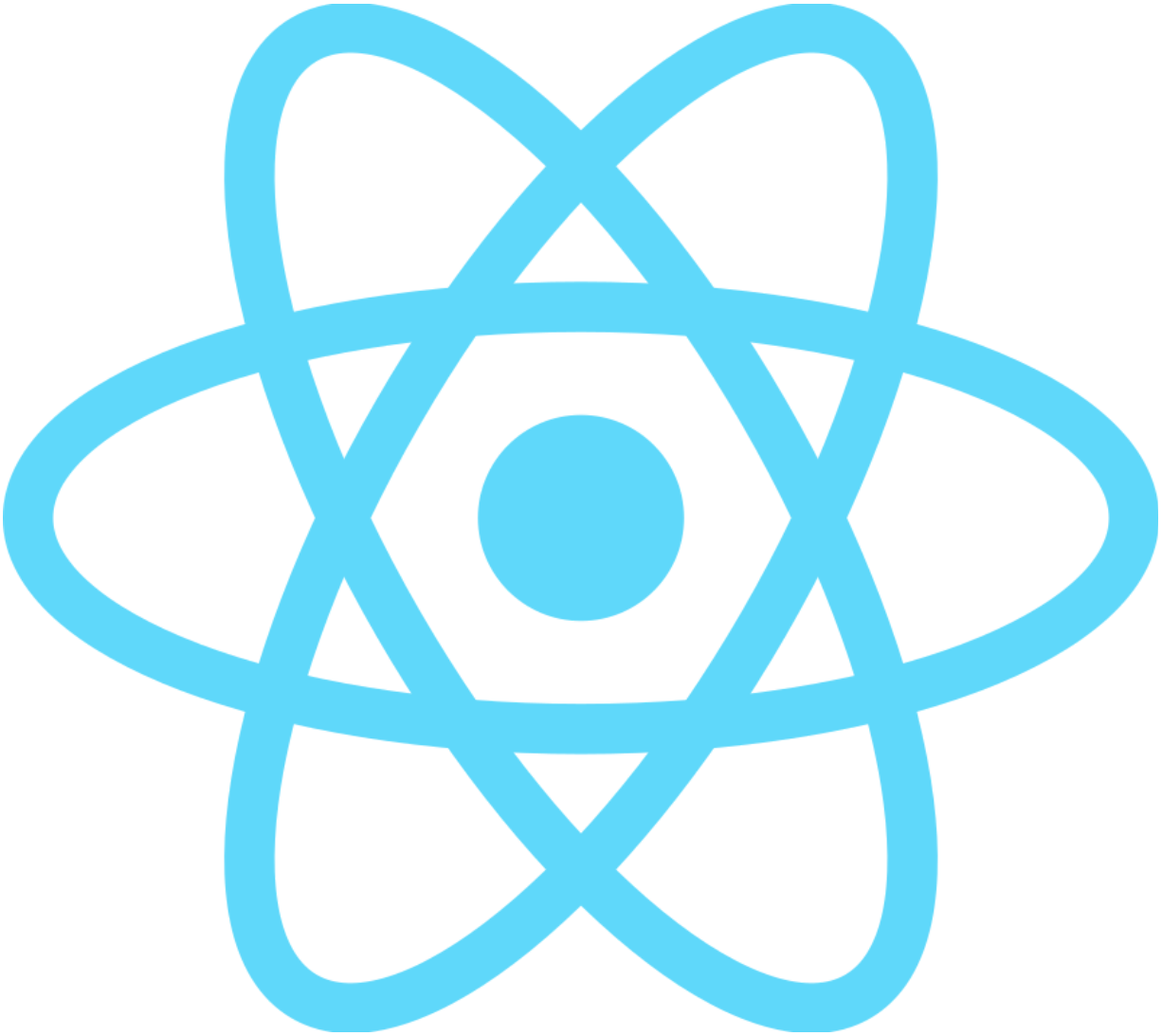
```
export default function Test({ title }) {
    const [items, setItems] = useState([]);

    useEffect(() => {
        axios.get('/api/items').then(resp => {
            setItems(resp.data);
        });
    }, []);
    return (
      <>
        <h1>{title}</h1>
        <ul className="items">
            {items.map(({ name, price, id }) => (
               <li key={id}>
                   { name } / { price }
                 </li>
            ))}
        </ul>
      </>
    );
}
```

La struttura risultante sarà:

```
<div class="wrapper">
    <h1>Risultati</h1>
    <ul class="items">
        <li>A / EUR 0.50</li>
        <li>B / EUR 1.99</li>
        <li>C / EUR 0.70</li>
    </ul>
</div>
```

Le proprietà possono anche essere riferimenti a funzioni, e questo è il modo con cui vengono gestiti in React gli eventi quando è necessario spostarne la gestione nei componenti genitori.

Applicazioni Correlate



Book Store

Un'applicazione in React con Node.js e Fastify come backend.
DockerDocker ComposeNode.jsJavaScriptFastifyReact