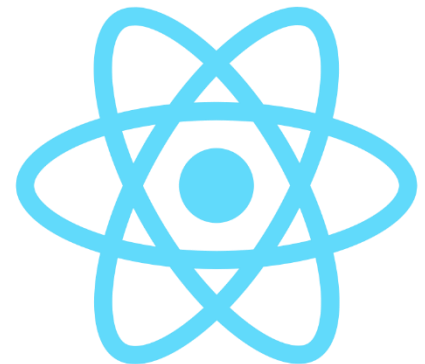


React

Riccardo Cattaneo



Eventi

Il primo gestore di eventi che andiamo a vedere è **onClick** che può essere applicato a qualsiasi tag anche se il più usato è il `<button>`. Ad esempio :

```
function App() {  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      {  
        persone.map((pers) => {  
          return <h1 key={pers.id}>{pers.cognome}</h1>;  
        })  
      }  
      <button onClick={ () => alert('inviato') }>INVIA</button>  
    </div>  
  );  
}  
  
export default App;
```



In questo primo esempio abbiamo scritto la nostra arrow function direttamente nel tag, ma possiamo anche creare una funzione con il codice e richiamarla al verificarsi dell'evento :

```
const invio = () => {  
  alert('sto inviando i dati');  
}  
  
function App() {  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      {  
        persone.map((pers) => {  
          return <h1 key={pers.id}>{pers.cognome}</h1>;  
        })  
      }  
      <button onClick={ invio }>INVIA</button>  
    </div>  
  );  
}
```

A red arrow originates from the `invio` variable in the `onClick={ invio }` prop of the `<button>` tag and points to the `invio` function definition at the top of the code block.

Se invece vogliamo passare dei parametri alla funzione sappiamo già come fare :

```
const invio = (nominativo) => {  
  alert(nominativo + ' sto inviando i dati');  
}  
  
function App() {  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      {  
        persone.map((pers) => {  
          return <h1 key={pers.id} onClick={ () => invio(pers.nome) }>{pers.cognome}</h1>;  
        })  
      }  
      <button >INVIA</button>  
    </div>  
  );  
}
```



The diagram consists of two red arrows. One arrow originates from the `invio` function definition in the first code block and points to the `invio` function call within the `persone.map` callback in the second code block. A second arrow originates from the `pers.nome` property access and points to the `invio` function call, indicating the argument being passed.

Un altro evento simile a onClick è **onMouseOver** per il quale valgono le stesse regole viste per l'onclick.

```
const invio = (nominativo,cognome) => {  
  console.log(nominativo + cognome + ' sto inviando i dati');  
}  
  
function App() {  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      {  
        persone.map((pers) => {  
          return <h1 key={pers.id} onMouseOver={ () => invio(pers.nome,pers.cognome) }>  
        })  
      }  
      <button >INVIA</button>  
    </div>  
  );  
}
```

A red arrow originates from the `onMouseOver={ () => invio(pers.nome,pers.cognome) }` attribute in the JSX, pointing to the `invio` function definition at the top of the code block. Another red arrow points from the `invio` function definition to the `console.log` statement inside its body.

Hooks

Prima di vedere cosa sono gli Hooks e come funzionano, andiamo a chiarire cosa fa react al cambio valore di una variabile. Per fare un esempio andiamo a creare un componente e al suo interno dichiariamo una variabile con valore 'ciao' ed una funzione che cambia il valore a tale variabile in 'arrivederci' ed all'interno del rendering creare un bottone che al verificarsi l'evento onClick richiama la funzione cambiaValore :

```
function App() {  
  
  let saluto = "ciao";  
  
  const cambioSaluto = () => {  
    saluto = "arrivederci";  
    console.log(saluto);  
  }  
  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      <h2>{saluto}</h2>  
      <button onClick={cambioSaluto}>Cambia</button>  
    </div>  
  );  
}
```

A diagram with three red arrows originating from a single point on the right side. One arrow points to the initial state 'ciao' in the 'let saluto' line. Another arrow points to the state update 'saluto = \"arrivederci\"' inside the 'cambioSaluto' function. The third arrow points to the 'onClick={cambioSaluto}' prop of the button in the JSX, illustrating how the state change is triggered by a user interaction.

Se vediamo il risultato all'interno della console possiamo constatare che effettivamente il valore della variabile è cambiato in memoria, ma non ha aggiornato il rendering del codice visualizzato a video.

Abbiamo bisogno quindi di «aggiornare» il nostro componente, per farlo ci vengono in aiuto gli Hooks e più precisamente **useState**.

useState

Quando creiamo un componente, non abbiamo a disposizione costruttori o altri metodi per inizializzare lo **stato** quindi dobbiamo utilizzare un'altra tecnica: gli **hooks**.

Nel caso specifico, il metodo **useState** è l'hook da utilizzare. Questo **metodo** accetta in input un valore che rappresenta il valore iniziale da dare a una variabile dello stato e restituisce in output un oggetto con una variabile che rappresenta lo stato e un metodo da invocare per modificare la variabile nello stato.

useState ha due funzionalità principali : ci permette di avere un render della variabile, in modo che lo stato viene mutato al mutare della variabile, ma soprattutto di tenere traccia dello stato che viene mutato.


Per prima cosa dobbiamo importare la funzione `useState` from `react` in questo modo :

```
import { useState } from 'react';  
import './App.css';  
  
function App() {  
  let saluto = "ciao";
```

La prima che andiamo a fare è stampare tramite `console.log` il valore di `useState` e possiamo vedere che si tratta di una funzione :

```
import { useState } from 'react';
import './App.css';

function App() {
  console.log(useState);
}
```



```
f useState(initialState) {
  var dispatcher = resolveDispatcher();
  return dispatcher.useState(initialState);
}
```

Navigated to <http://localhost:3000/>

Se proviamo a passare alla funzione `useState` un valore di qualsiasi tipo (un oggetto, una stringa, un numero o qualsiasi tipo) vediamo che ci ritorna un array composto dal valore passato in ingresso ed in più una funzione :

```
import { useState } from 'react';
import './App.css';

function App() {

  const value1 = useState()[0];
  const value2 = useState()[1];

  console.log(value1,value2);
}
```

React

```
undefined f dispatchAction(fiber, queue, action) {
  {
    if (typeof arguments[3] === 'function') {
      error("State updates from the useState() and useReducer() Hooks don't support the " + 'secon...
```


Dopo questo primo esempio andiamo a creare un **useState** reale, riproponendo lo stesso esempio di prima, e cioè creare un bottone che chiama una funzione per cambiare valore alla variabile :

```
function App() {  
  
  const salutami = useState('buongiorno');  
  
  console.log('primo ' + salutami[0]);  
  console.log('secondo ' + salutami[1]);  
  
  const cambioSaluto = () => {  
    salutami[1]("arrivederci");  
    console.log(salutami[0]);  
  }  
  
  return (  
    <div className="App">  
      <h1>Componente Principale</h1>  
      <h2>{salutami[0]}</h2>  
      <button onClick={cambioSaluto}>Cambia</button>  
    </div>  
  );  
}
```

Destrutturazione

```
const UseBase = () => {  
  // const value = useState()[0];  
  // const handler = useState()[1];  
  // console.log(value, handler);  
  const [titolo, setTitolo] = useState("Hello World!!");  
  const cambiaTitolo = () => {  
    if (titolo === "Hello World!!") {  
      setTitolo("React Magic!");  
    } else {  
      setTitolo("Hello World!!");  
    }  
  };  
  return (  

```

A diagram with two red arrows. One arrow starts from the opening square bracket of the array destructuring [titolo, setTitolo] in the useState call and points to the first element 'titolo'. The second arrow starts from the closing square bracket of the same array and points to the second element 'setTitolo'.

Alcune Regole

- La prima regola da ricordare è che tutti gli Hook **DEVONO** iniziare con **use**, anche quelli che creeremo noi.
- Il componente in cui usiamo gli Hook **DEVE** avere il nome con la prima lettera **maiuscola**.
- Gli Hook **DEVONO** essere chiamati **all'interno** del nostro componente

useState con Array

Prima di vedere come si comporta useState con un array andiamo a creare una cartella dove creiamo un file chiamato data.js ed andiamo a mettere un array di oggetti con due campi : id e nome e popoliamola a piacimento con qualche nominativo (simulando la chiamata ad un database). Ricordiamo anche di importare useState dalla libreria di react :


```
src > data > JS data.js > [🔍] anagrafica
```

```
1  export const anagrafica = [  
2      {id : 1, nome : "Mario"},  
3      {id : 2, nome : "Luca"},  
4      {id : 3, nome : "Anna"},  
5      {id : 4, nome : "Giuseppe"},  
6      {id : 5, nome : "Francesco"}  
7  ]
```

```
import { useState } from 'react';  
import './App.css';
```

Se ora proviamo a passare alla funzione useState l'array anagrafica appena importata possiamo verificare che il risultato è sempre lo stesso, la funzione ci torna l'oggetto passato in ingresso ed una funzione :

```
import { useState } from 'react';
import { anagrafica } from '../data/data';
import './App.css';

function App() {

  const persone = useState(anagrafica);

  console.log('primo ' + persone[0]);
  console.log('secondo ' + persone[1]);
```

Destrutturazione

Un modo alternativo di gestire il ritorno di una funzione, se il ritorno è composto da più oggetti è possibile «destrutturarli» in questo modo, ma è solo per comodità :

```
const persone = useState(anagrafica);  
// destrutturazione  
const [anag, setAnag] = useState(anagrafica);  
  
console.log('primo ' + anag);  
console.log('secondo ' + setAnag);  
//console.log('primo ' + persone[0]);  
//console.log('secondo ' + persone[1]);
```

Arrivati a questo punto abbiamo un array che se vogliamo stamparlo a video già sappiamo come fare.... Usando la funzione map in questo modo : l'unica differenza rispetto a prima che aggiungiamo un bottone vicino ad ogni oggetto ...

```
return (  
  <div className="App">  
    <h1>Componente Principale</h1>  
  
    {anag.map( el => {  
      const {id, nome} = el;  
      return(  
        <div key={id}>  
          <span>{nome}</span>  
          <button>Elimina</button>  
        </div>  
      )  
    })}  
  )  
}
```

Il nostro obiettivo è quello di eliminare una voce dall'array nel momento in cui clicchiamo sul corrispondente bottone Elimina.

Per farlo andiamo a creare una funzione che richiameremo al click, e questa funzione attraverso la funzione filter va a selezionare gli oggetti in base ad una «condizione» passata in ingresso come parametro :

```
const eliminaoggetto = (id) => {
  let nuoviOggetti = anag.filter( el => el.id !== id);
  setAnag(nuoviOggetti);
}

return (
  <div className="App">
    <h1>Componente Principale</h1>

    {anag.map( el => {
      const {id, nome} = el;
      return(
        <div key={id}>
          <span>{nome}</span>
          <button onClick={() => eliminaoggetto(id)}>Elimina</button>
        </div>
      )
    })}

  </div>
);
```

useState con Oggetto

Se usiamo un oggetto con useState non ci cambia nulla, dobbiamo solo prestare attenzione ad un particolare. Supponiamo di avere un oggetto persona con nome, cognome ed età, e tramite un evento vado a cambiare il valore di un solo elemento dell'oggetto, cosa succede ? Proviamo :

```
const [persona, setPersona] = useState({
  nome : 'Riccardo',
  cognome : 'Cattaneo',
  eta : 30,
});

const compleanno = () => {
  let anni = persona.eta + 1;
  setPersona({
    eta : anni
  });
}

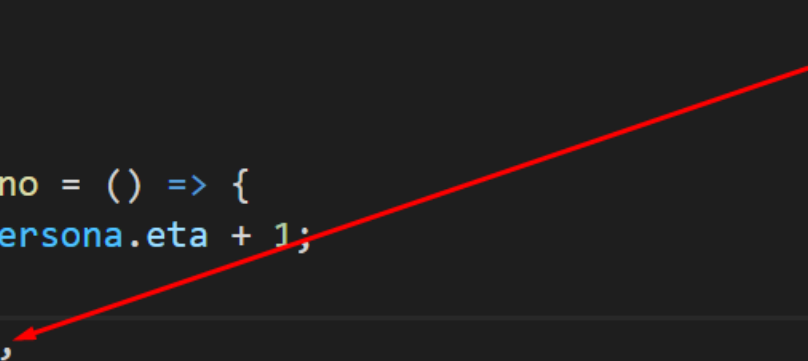
return (
  <div className="App">
    <h1>Componente Principale</h1>
    <h3>{persona.nome}</h3>
    <h3>{persona.cognome}</h3>
    <h3>{persona.eta}</h3>
    <button onClick={compleanno}>Compleanno</button>
  </div>
);
}
```


Se proviamo questo codice ci accorgiamo di due cose... la prima è che «funziona» 😊 la seconda è che, se aggiorniamo un solo dato del nostro oggetto, il resto lo perdiamo, quindi abbiamo il problema di mantenere in qualche modo l'oggetto originale, e qui ci viene in aiuto lo **Spread Operator**, ricordate a cosa serve ? Serve a passare in ingresso ad una funzione o ad un componente un intero oggetto

```
const [persona, setPersona] = useState({
  nome : 'Riccardo',
  cognome : 'Cattaneo',
  eta : 30,
});

const compleanno = () => {
  let anni = persona.eta + 1;
  setPersona({
    ...persona,
    eta : anni
  });
}

return (
  <div className="App">
    <h1>Componente Principale</h1>
    <h3>{persona.nome}</h3>
    <h3>{persona.cognome}</h3>
    <h3>{persona.eta}</h3>
    <button onClick={compleanno}>Compleanno</button>
  </div>
);
```



Esercizio

Creare un componente che rappresenta per la mia applicazione un contatore utilizzando useState con valore di default uguale a zero. Sotto al contatore andiamo a creare due bottoni, uno «diminuisci» ed uno «aumenta» che al loro click andranno ad aumentare e a diminuire il valore del contatore.

Utilizziamo bootstrap per rendere gradevole il nostro componente (a nostro piacimento)