JavaScript ReactJs

Introduzione a Reactjs 2

Giovanni Iovino: giovanni.iovino@intecs.it

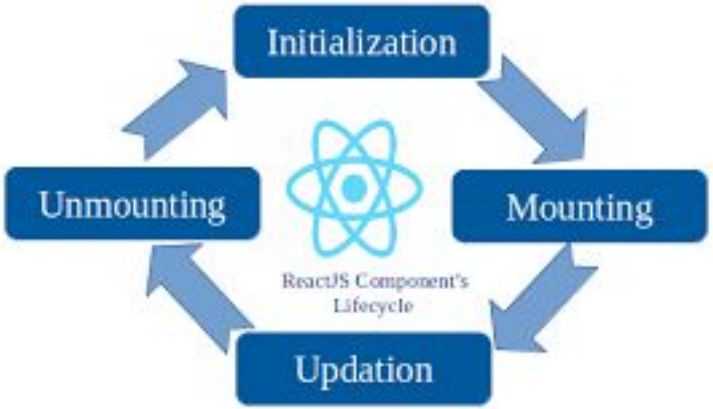
Contenuto:

Introduzione a React: Ciclo di vita dei componenti / propagazione props

React - Ciclo di vita dei componenti

Ogni componente React ha un ciclo di vita diviso in diverse fasi che possono essere monitorate:

- Inizializzazione: Quando il componente viene istanziato
- Mounting: Fase in cui il JSX del componente viene caricato nel DOM
- Updating: fase che si ripete tutte le volte che il componente viene aggiornato a causa di un cambiamento di stato o di un cambiamento delle props
- Unmounting: Rimozione del componente dal DOM



React - Mounting

Durante la fase di mounting vengono chiamate i seguenti metodi della classe React.Component:

- **constructor()** - Costruttore del componente. Usato per inizializzare lo stato e le altre proprietà del componente.
- **static getDerivedStateFromProps(props, state)** - Funzione che viene chiamata in fase di mounting e ogni volta che cambiano le props. Usa le nuove props e lo stato attuale come argomenti e restituisce il nuovo stato del componente. Può essere utilizzata per aggiornare lo stato al cambio delle props, il suo uso andrebbe evitato in quanto ci sono design pattern migliori per aggiornare lo stato al cambio delle peops.
- **render()** - Funzione che si occupa di restituire il JSX che determina cosa inserire nel DOM
- **componentDidMount()** - Funzione chiamata dopo che il componente è inserito nel DOM. Utilizzata per effettuare operazioni che per essere eseguite richiedono che il DOM sia stato aggiornato. Finché il DOM non è aggiornato non è possibile accedere agli elementi DOM del componente.

Nota: render() è l'unico metodo che un componente deve implementare obbligatoriamente.

```
class ExampleComponent extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
      text: undefined,
      message: ""
    };
  }

  componentDidMount() {
    this.timerID = setInterval(
      () =>{
        const counter = this.state.counter;
        this.setState({counter: counter+1});
      },
      5000
    );
    this.setState({message: "mounted"})
  }

  static getDerivedStateFromProps(props, state){
    /*if(props.start==="mille"){ //Cosa è sbagliato?
      state.counter=1000;
    }*/
    return state;
  }

  componentDidUpdate(){
    if(this.state.message==="mounted"){
      this.setState({message: "updated"})
    }
  }
  componentWillUnmount() {
    this.clearInterval(this.timerID);
  }

  shouldComponentUpdate(nextProps, nextState){
    //if(nextState.counter%2==0) return false;
    return true;
  }

  getSnapshotBeforeUpdate(prevProps, prevState){
  }

  render() {
    return (
      <div>
        <h1>Example component</h1>
        <div>Counter: {this.state.counter}.</div>
        <div>Message: {this.state.message}.</div>
      </div>
    );
  }
}
```

React - Updating

Durante la fase di aggiornamento vengono chiamate i seguenti metodi della classe React.Component:

- **static getDerivedStateFromProps(props, state) -** Come descritto nella fase di mounting.
- **shouldComponentUpdate(nextProps, nextState) -** Metodo usato raramente per ottimizzare le performance. Internamente si possono usare i riferimenti a this.props e this.state e confrontarli con gli argomenti nextProps e nextState per decidere se re-renderizzare il componente. Per farlo la funzione deve restituire true o false.
- **render() -** Come descritto nella fase di mounting.
- **getSnapshotBeforeUpdate(prevProps, prevState) -** Usato raramente. Chiamato dopo il render ma prima che i cambiamenti siano consolidati. Può essere usato per interrogare il DOM prima che sia modificato dall'aggiornamento.
- **componentDidUpdate() -** Chiamata alla fine della fase di aggiornamento dopo che tutte le modifiche al DOM sono state applicate. Utilizzato per effettuare operazioni dopo che il DOM è stato aggiornato. Non è chiamato al primo rendering dove invece viene chiamato **componentDidMount()**

```
class ExampleComponent extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
      text: undefined,
      message: ""
    };
  }

  componentDidMount() {
    this.timerID = setInterval(
      () =>{
        const counter = this.state.counter;
        this.setState({counter: counter+1});
      },
      5000
    );
    this.setState({message: "mounted"})
  }

  static getDerivedStateFromProps(props, state){
    /*if(props.start==="mille"){ //Cosa è sbagliato?
      state.counter=1000;
    }*/
    return state;
  }

  componentDidUpdate(){
    if(this.state.message==="mounted"){
      this.setState({message: "updated"})
    }
  }
  componentWillUnmount() {
    this.clearInterval(this.timerID);
  }

  shouldComponentUpdate(nextProps, nextState){
    //if(nextState.counter%2==0) return false;
    return true;
  }

  getSnapshotBeforeUpdate(prevProps, prevState){
  }

  render() {
    return (
      <div>
        <h1>Example component</h1>
        <div>Counter: {this.state.counter}.</div>
        <div>Message: {this.state.message}.</div>
      </div>
    );
  }
}
```

React - Unmounting

Durante la fase di unmounting vengono chiamate i seguenti metodi della classe React.Component:

- **componentWillUnmount()** - Chiamato subito prima che il componente venga distrutto. Usato per effettuare il rilascio delle risorse che potrebbero sopravvivere alla distruzione del componente (es. timers)

```
class ExampleComponent extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      counter: 0,
      text: undefined,
      message: ""
    };
  }

  componentDidMount() {
    this.timerID = setInterval(
      () =>{
        const counter = this.state.counter;
        this.setState({counter: counter+1});
      },
      5000
    );
    this.setState({message: "mounted"})
  }

  static getDerivedStateFromProps(props, state){
    /*if(props.start==="mille"){ //Cosa è sbagliato?
      state.counter=1000;
    }*/
    return state;
  }

  componentDidUpdate(){
    if(this.state.message==="mounted"){
      this.setState({message: "updated"})
    }
  }
  componentWillUnmount() {
    this.clearInterval(this.timerID);
  }

  shouldComponentUpdate(nextProps, nextState){
    //if(nextState.counter%2==0) return false;
    return true;
  }

  getSnapshotBeforeUpdate(prevProps, prevState){
  }

  render() {
    return (
      <div>
        <h1>Example component</h1>
        <div>Counter: {this.state.counter}.</div>
        <div>Message: {this.state.message}.</div>
      </div>
    );
  }
}
```

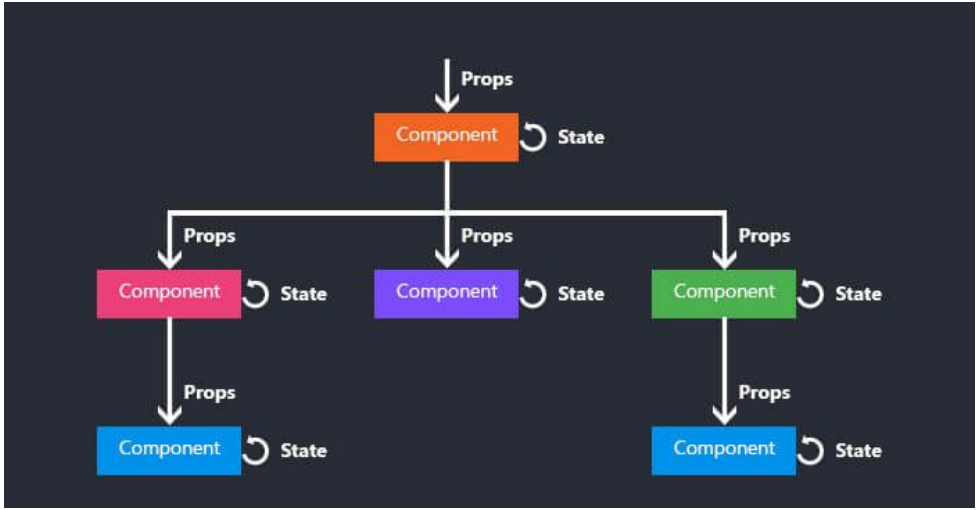
React - State

Alcune informazioni che è necessario conoscere a proposito dello stato di un componente:

- A parte che nel costruttore, non va mai cambiato lo stato direttamente. Non funzionerebbe. Si deve utilizzare **setState({..})**
- I cambiamenti di stato avvengono tramite merge. Non è necessario passare a **setState** tutte le proprietà dello stato, ma solo quelle che devono essere aggiornate.
- Il cambiamento di stato non è sincrono. Quando si invoca il setState non è detto che lo stato sia aggiornato immediatamente. Se c'è la necessità di sapere quando le modifiche della **setState** sono state applicate si può passare una callback come secondo argomento, che verrà chiamata ad aggiornamento avvenuto: **setState({...}, callback)**
- I cambiamenti di stato non andrebbero fatti prima che finisca la fase di mounting e in componentWillUnmount(), dato che un cambiamento di stato implica un update e richiede che il componente sia stato inserito nel DOM.

React - Propagazione delle props

- Normalmente i componenti non hanno accesso ai metodi degli altri componenti.
- Per far interagire i componenti tra loro si utilizzano le props, che si propagano a cascata. Questo permette di abilitare interazioni di tipo parent<->child
- Nei rari casi in cui è necessario accedere direttamente ad un'istanza di un componente si possono usare le **refs:** https://it.reactjs.org/docs/forwarding-refs.html



Altre tecniche per gestire gli stati tra più componenti (further reading):

https://react-redux.js.org/