Metodi Funzionali per la Gestione degli Array in JavaScript

Introduzione

Cosa sono i metodi funzionali degli array?

- **Definizione**: Metodi che permettono di manipolare e processare gli array in modo dichiarativo e funzionale.
- **Vantaggi**:
  - Codice più pulito e leggibile.
  - Evita l'uso esplicito di cicli.
  - Favorisce l'immutabilità dei dati.

Metodo `forEach`

Cosa fa?

- Esegue una funzione callback su ogni elemento dell'array.

Sintassi

```
array.forEach((elemento, indice, array) => {
  // Corpo della funzione
});
```

Metodo `forEach`

Esempio

```
const numeri = [1, 2, 3];
numeri.forEach(numero => {
  console.log(numero * 2);
});
// Output: 2, 4, 6
```

Metodo `map`

Cosa fa?

- Crea un nuovo array applicando una funzione a ciascun elemento dell'array originale.

Sintassi

```
const nuovoArray = array.map((elemento, indice, array) => {
  return nuovoElemento;
});
```

Metodo `map`

Esempio

```
const numeri = [1, 2, 3];
const doppi = numeri.map(numero => numero * 2);
console.log(doppi);
// Output: [2, 4, 6]
```

Metodo `filter`

Cosa fa?

- Crea un nuovo array con tutti gli elementi che soddisfano una determinata condizione.

Sintassi

```
const nuovoArray = array.filter((elemento, indice, array) => {
  return condizione;
});
```

Metodo `filter`

Esempio

```
const numeri = [1, 2, 3, 4];
const pari = numeri.filter(numero => numero % 2 === 0);
console.log(pari);
// Output: [2, 4]
```

Metodo `reduce`

Cosa fa?

- Riduce l'array a un singolo valore applicando una funzione accumulatrice su ogni elemento.

Sintassi

```
const risultato = array.reduce((accumulatore, elemento, indice, array) => {
  return nuovoAccumulatore;
}, valoreIniziale);
```

Metodo `reduce`

Esempio

```
const numeri = [1, 2, 3, 4];
const somma = numeri.reduce((acc, numero) => acc + numero, 0);
console.log(somma);
// Output: 10
```

Metodo `find`

Cosa fa?

- Restituisce il primo elemento dell'array che soddisfa una determinata condizione.

Sintassi

```
const elemento = array.find((elemento, indice, array) => {
  return condizione;
});
```

Esempio

```
const numeri = [1, 2, 3, 4];
const primoPari = numeri.find(numero => numero % 2 === 0);
console.log(primoPari);
// Output: 2
```

Metodo `some`

Cosa fa?

- Verifica se almeno un elemento dell'array soddisfa una determinata condizione. Restituisce un valore booleano.

Sintassi

```
const esiste = array.some((elemento, indice, array) => {
  return condizione;
});
```

Metodo `some`

Esempio

```
const numeri = [1, 3, 5];
const pari = numeri.some(numero => numero % 2 === 0);
console.log(pari);
// Output: false
```

Metodo `every`

Cosa fa?

- Verifica se tutti gli elementi dell'array soddisfano una determinata condizione. Restituisce un valore booleano.

Sintassi

```
const tutti = array.every((elemento, indice, array) => {
  return condizione;
});
```

Metodo `every`

Esempio

```
const numeri = [2, 4, 6];
const tuttiPari = numeri.every(numero => numero % 2 === 0);
console.log(tuttiPari);
// Output: true
```

Operatore Spread

Cosa fa?

- Consente di espandere un array o un oggetto in un altro contesto.

Esempio

```
const numeri = [1, 2, 3];
const nuoviNumeri = [...numeri, 4, 5];
console.log(nuoviNumeri);
// Output: [1, 2, 3, 4, 5]
```

Operatore Rest

Cosa fa?

- Permette di raccogliere più valori in un array.

Esempio

```
function somma(...numeri) {
  return numeri.reduce((acc, num) => acc + num, 0);
}
console.log(somma(1, 2, 3, 4));
// Output: 10
```

Destrutturazione di Array

Cosa fa?

- Permette di estrarre elementi di un array in variabili separate.

Esempio

```
const numeri = [1, 2, 3, 4];
const [primo, secondo, ...others] = numeri;
console.log(primo, secondo, others);
// Output: 1, 2, [3, 4]
```

Destrutturazione di Oggetti

Cosa fa?

- Permette di estrarre proprietà di un oggetto in variabili separate.

Esempio

```
const persona = { nome: "Mario", eta: 30 };
const { nome, eta } = persona;
console.log(nome, eta);
// Output: Mario, 30
```

Conclusione

- I metodi funzionali degli array in JavaScript offrono un modo potente e conciso per manipolare e processare i dati.
- Utilizzando `forEach`, `map`, `filter`, `reduce`, `find`, `some` ed `every`, è possibile scrivere codice più leggibile e mantenibile.
- L'operatore **spread** e **rest** semplificano la gestione di array e parametri.
- La **destrutturazione** di array e oggetti migliora la leggibilità e l'accesso ai dati.

Domande?

Grazie per l'attenzione!


Q&A Icon

Esempi pratici o chiarimenti?