JavaScript: Strutture Dati Flessibili - Gli Oggetti

Introduzione agli Oggetti

Gli oggetti in JavaScript sono un tipo di dato che ci permette di memorizzare una collezione di proprietà e metodi. Rappresentano uno dei concetti più potenti e flessibili del linguaggio, consentendoci di modellare entità del mondo reale attraverso il codice.

Un oggetto può essere immaginato come un contenitore che raccoglie dati correlati e funzionalità in un'unica struttura organizzata. A differenza degli array, che utilizzano indici numerici per accedere agli elementi, gli oggetti utilizzano chiavi (nomi di proprietà) per organizzare e accedere ai dati.

Creazione di Oggetti

La sintassi base per creare un oggetto in JavaScript utilizza le parentesi graffe `{}`:

```javascript
let nomeOggetto = {
  nomeProprietà: valoreProprietà,
  nomeProprietà: valoreProprietà,
  // ...
};
```

Ecco alcuni esempi di oggetti:

```javascript
// Un oggetto semplice
let suDiMe = {
  città: 'Roma, IT',
  capelli: 'castani'
};

// Un oggetto più complesso con valori di diversi tipi
let mioGatto = {
  età: 8,
  colorePelo: 'grigio',
  piace: ['acqua', 'latte', 'carne'],
  compleanno: {'mese': 7, 'giorno': 17, anno: 1994}
};
```

Come puoi vedere nel secondo esempio, le proprietà di un oggetto possono contenere valori di qualsiasi tipo, inclusi array e altri oggetti. Questa caratteristica rende gli oggetti estremamente

versatili per rappresentare strutture dati complesse.

Accesso alle Proprietà degli Oggetti

Ci sono due modi principali per accedere alle proprietà di un oggetto in JavaScript:

1. Notazione a punto (Dot Notation)

```javascript
let suDiMe = {
  città: 'Roma, IT',
  capelli: 'castani'
};

// Accesso alle proprietà usando la notazione a punto
let miaCittà = suDiMe.città;  // 'Roma, IT'

// Le proprietà non esistenti restituiranno undefined
let mioGenere = suDiMe.genere;  // undefined
```

La notazione a punto è la più comune e leggibile quando conosciamo in anticipo il nome della proprietà a cui vogliamo accedere.

2. Notazione a parentesi quadre (Bracket Notation)

```javascript
let suDiMe = {
  città: 'Roma, IT',
  capelli: 'castani'
};

// Accesso alle proprietà usando la notazione a parentesi quadre
let mieiCapelli = suDiMe['capelli'];  // 'castani'

// Le proprietà non esistenti restituiranno undefined
let mioGenere = suDiMe['genere'];  // undefined

// Le proprietà possono anche essere accedute con chiavi variabili
let miaProprietà = 'capelli';
let mieiCapelli = suDiMe[miaProprietà];  // 'castani'
```

La notazione a parentesi quadre è particolarmente utile quando:

- Il nome della proprietà è memorizzato in una variabile
- Il nome della proprietà contiene caratteri speciali o spazi

- Il nome della proprietà viene determinato dinamicamente durante l'esecuzione

Modifica degli Oggetti

Gli oggetti in JavaScript sono mutabili, il che significa che possiamo modificarli dopo la loro creazione. Possiamo utilizzare sia la notazione a punto che quella a parentesi quadre insieme all'operatore di assegnazione per modificare gli oggetti:

```javascript
let suDiMe = {
  città: 'Roma, IT',
  capelli: 'castani'
};

// Modificare proprietà esistenti
suDiMe.capelli = 'blu';

// Aggiungere nuove proprietà
suDiMe.genere = 'femminile';

// Eliminare proprietà
delete suDiMe.genere;
```

Questa mutabilità rende gli oggetti molto flessibili, permettendoci di aggiungere, modificare o rimuovere proprietà in qualsiasi momento.

Array di Oggetti

Poiché gli array possono contenere qualsiasi tipo di dato, possono anche contenere oggetti. Gli array di oggetti sono una struttura dati estremamente utile in JavaScript, spesso utilizzata per rappresentare collezioni di entità simili:

```javascript
// Array di oggetti
let mieiGatti = [
  {nome: 'Cleo',
   età: 8},
  {nome: 'Simba',
   età: 1}
];

// Iterazione attraverso un array di oggetti
for (let i = 0; i < mieiGatti.length; i++) {
  let mioGatto = mieiGatti[i];
  console.log(mioGatto.nome + ' ha ' + mioGatto.età + ' anni.');
}
```

Questa è una struttura dati molto comune nelle applicazioni web, spesso utilizzata per rappresentare liste di utenti, prodotti, messaggi, o qualsiasi altra collezione di elementi con proprietà simili.

Oggetti come Argomenti di Funzioni

Come altri tipi di dati, gli oggetti possono essere passati come argomenti alle funzioni:

```javascript
let mioGatto = {
  età: 8,
  colorePelo: 'grigio',
  piace: ['acqua', 'latte', 'carne'],
  compleanno: {'mese': 7, 'giorno': 17, 'anno': 1994}
};

function descriviGatto(gatto) {
  console.log('Questo gatto ha ' + gatto.età + ' anni con pelo ' + gatto.colorePelo + '.');
}

descriviGatto(mioGatto);  // "Questo gatto ha 8 anni con pelo grigio."
```

Passare oggetti come argomenti è un modo potente per fornire molte informazioni correlate a una funzione, invece di passare ogni valore separatamente.

Metodi degli Oggetti

Le proprietà di un oggetto possono anche essere funzioni. Le funzioni all'interno di un oggetto sono chiamate "metodi":

javascript

```javascript
let mioGatto = {
  età: 8,
  colorePelo: 'grigio',
  miagola: function() {
    console.log('miaooo');
  },
  mangia: function(cibo) {
    console.log('Yum, adoro ' + cibo);
  },
  dormi: function(minuti) {
    for (let i = 0; i < minuti; i++) {
      console.log('z');
    }
  }
};

// Invocare i metodi di un oggetto usando la notazione a punto
mioGatto.miagola();  // "miaooo"
mioGatto.mangia('cibo per gatti');  // "Yum, adoro cibo per gatti"
mioGatto.dormi(3);  // Stampa "z" tre volte
```

I metodi permettono agli oggetti di avere comportamenti oltre a semplici dati, rendendo possibile creare modelli più completi e autocontenuti.

La parola chiave `this` nei metodi

Quando un metodo viene invocato, la parola chiave `this` si riferisce all'oggetto che contiene il metodo:

javascript

```javascript
let mioGatto = {
  età: 8,
  nome: 'Cleo',
  diNome: function() {
    console.log('Mi chiamo ' + this.nome);
  }
};

mioGatto.diNome();  // "Mi chiamo Cleo"
```

La parola chiave `this` è fondamentale nei metodi degli oggetti perché permette a un metodo di accedere e manipolare i dati dell'oggetto stesso.

Object.keys()

Il metodo `Object.keys()` elenca tutti i nomi delle proprietà di un oggetto in un array:

```javascript
let mioGatto = {
  età: 8,
  colorePelo: 'grigio',
  miagola: function() {
    console.log('miaooo');
  },
  dormi: function(minuti) {
    for (let i = 0; i < minuti; i++) {
      console.log('z');
    }
  }
};

console.log(Object.keys(mioGatto));  // ['età', 'colorePelo', 'miagola', 'dormi']
```

Questo metodo è molto utile per iterare dinamicamente sulle proprietà di un oggetto quando non conosciamo in anticipo tutti i nomi delle proprietà.

Oggetti Incorporati (Built-in Objects)

JavaScript ha molti oggetti utili incorporati nel linguaggio:

- **Array**: Per lavorare con collezioni ordinate di elementi
  - `Array.isArray()`: Verifica se un valore è un array
- **Number**: Per lavorare con valori numerici
  - `Number()`: Converte un valore in un numero
  - `Number.parseInt()`: Converte una stringa in un numero intero
  - `Number.parseFloat()`: Converte una stringa in un numero decimale
- **Date**: Per lavorare con date e orari
  - `Date.UTC()`: Restituisce un timestamp per una data UTC specifica
  - `Date.now()`: Restituisce il timestamp corrente
  - `Date.parse()`: Converte una stringa di data in un timestamp
- **Math**: Fornisce funzioni matematiche avanzate
  - Contiene molte funzioni utili per calcoli matematici

Questi oggetti incorporati estendono le funzionalità base del linguaggio e forniscono strumenti potenti per lavorare con vari tipi di dati e operazioni.

Copia per valore vs. copia per riferimento

In JavaScript, comprendere la differenza tra copia per valore e copia per riferimento è fondamentale:

Copia per valore (primitivi)

I tipi di dati primitivi (numeri, stringhe, booleani, null, undefined) vengono copiati per valore:

```javascript
// Copia per valore (primitivi)
let a = 5;
let b = a;  // b è ora una copia di a
b = 10;  // modificare b non influisce su a
console.log(a);  // output: 5
console.log(b);  // output: 10
```

Quando assegniamo un valore primitivo da una variabile a un'altra, viene creata una copia indipendente del valore.

Copia per riferimento (oggetti)

Gli oggetti (inclusi array e funzioni) vengono copiati per riferimento:

```javascript
// Copia per riferimento (oggetti)
let obj1 = { nome: 'Giovanni' };
let obj2 = obj1;  // obj2 è ora un riferimento a obj1
obj2.nome = 'Gianna';  // modificare obj2 influenzerà anche obj1
console.log(obj1.nome);  // output: Gianna
console.log(obj2.nome);  // output: Gianna
```

Quando assegniamo un oggetto da una variabile a un'altra, entrambe le variabili fanno riferimento allo stesso oggetto in memoria. Modificare una variabile influisce sull'altra perché puntano allo stesso oggetto.

Passaggio di parametri: valore vs. riferimento

Il comportamento di copia per valore o per riferimento si estende anche al passaggio di parametri alle funzioni:

```javascript
function aggiornaValori(num, obj, arr) {
  num = 10;
  obj.nome = 'Gianna';
  arr.push(4);
}

let a = 5;
let persona = { nome: 'Giovanni' };
let numeri = [1, 2, 3];

aggiornaValori(a, persona, numeri);

console.log(a);  // output: 5 - invariato perché passato per valore
console.log(persona.nome);  // output: Gianna - cambiato perché passato per riferimento
console.log(numeri);  // output: [1, 2, 3, 4] - cambiato perché passato per riferimento
```

Questo comportamento è importante da comprendere quando si lavora con funzioni che modificano i loro parametri:

- I parametri di tipo primitivo non possono essere modificati in modo permanente dalla funzione
- I parametri di tipo oggetto possono essere modificati dalla funzione

Invocazione con notazione a parentesi quadre

I metodi degli oggetti possono essere invocati non solo con la notazione a punto, ma anche con la notazione a parentesi quadre:

```javascript
let auto = {
  avviaMotore: function () {
    console.log('Motore avviato.');
  }
};

// Invocare il metodo usando la notazione a parentesi quadre
auto['avviaMotore']();  // "Motore avviato."
```

Questo è particolarmente utile quando il nome del metodo è memorizzato in una variabile o determinato dinamicamente:

javascript

```javascript
let auto = {
  avviaMotore: function (tipoCarburante) {
    console.log('Motore avviato. Tipo di carburante: ' + tipoCarburante);
  },
  arrestaMotore: function () {
    console.log('Motore arrestato.');
  }
};

// Invocare i metodi usando la notazione a parentesi quadre
auto['avviaMotore']('benzina');  // "Motore avviato. Tipo di carburante: benzina"
auto['arrestaMotore']();  // "Motore arrestato."

// Invocare i metodi usando la notazione a punto (più comune)
auto.avviaMotore('benzina');  // "Motore avviato. Tipo di carburante: benzina"
auto.arrestaMotore();  // "Motore arrestato."
```

Oggetti Immutabili

Oggetti costanti ma mutabili

In JavaScript, dichiarare un oggetto con `const` significa che il riferimento all'oggetto non può essere riassegnato, ma le proprietà dell'oggetto possono ancora essere modificate:

javascript

```javascript
// mioGatto è una costante, ma è mutabile quindi le sue proprietà possono cambiare
const mioGatto = {
  nome: 'Cleo',
  età: 8
};

mioGatto.nome = 'Adam';
console.log(mioGatto.nome);  // 'Adam' non 'Cleo'

mioGatto = {};  // TypeError: Assignment to constant variable
```

Rendere gli oggetti effettivamente immutabili

Per rendere un oggetto veramente immutabile (impedire qualsiasi modifica alle sue proprietà), possiamo usare `Object.freeze()`:

```javascript
// Gli oggetti mutabili possono essere modificati nel tempo, mentre quelli immutabili non
const mioGatto = {
  nome: 'Cleo',
  età: 8
};

Object.freeze(mioGatto);
mioGatto.nome = 'Sam';
console.log(mioGatto.nome);  // 'Cleo' non 'Sam'
```

`Object.freeze()` impedisce l'aggiunta di nuove proprietà, la rimozione di quelle esistenti e la modifica dei valori delle proprietà esistenti. Tuttavia, è importante notare che `Object.freeze()` esegue un "congelamento superficiale" (shallow freeze) - se un oggetto contiene altri oggetti, questi oggetti annidati non saranno immutabili a meno che non vengano congelati separatamente.

Considerazioni finali

Gli oggetti in JavaScript sono una struttura dati potente e versatile che ci permette di organizzare dati e comportamenti correlati in un'unica entità. Le loro caratteristiche chiave includono:

1. **Flessibilità**: Possono contenere proprietà di qualsiasi tipo di dato, inclusi altri oggetti e funzioni
2. **Accesso dinamico**: Le proprietà possono essere accedute e modificate in vari modi
3. **Comportamento tramite metodi**: Gli oggetti possono avere comportamenti tramite metodi
4. **Riferimenti vs valori**: Comprendere la differenza tra copia per valore e copia per riferimento è fondamentale

La padronanza degli oggetti è essenziale per la programmazione JavaScript efficace e la creazione di applicazioni web robuste. Gli oggetti formano la base di molti concetti avanzati in JavaScript, inclusa la programmazione orientata agli oggetti e i pattern di progettazione comuni utilizzati nello sviluppo web moderno.

Padroneggiando gli oggetti, si apre la strada a una programmazione JavaScript più strutturata, organizzata e potente, che può affrontare problemi complessi in modo elegante ed efficiente.