

THIS in Javascript

Stefano Cherio e Nicolò Galizia

Il concetto di **this** in Javascript

Introduzione


- **This** è una parola chiave in JavaScript che rappresenta il **contesto di esecuzione di una funzione**.
- Il valore di **this** **cambia dinamicamente** in base a come e dove viene chiamata una funzione.
- Può essere difficile da capire, specialmente perché il suo **comportamento cambia** a seconda del contesto.

Regole base di this

Contesto globale

Quando **this** è utilizzato al di fuori di qualsiasi funzione o oggetto, in un **contesto globale**, si riferisce all'**oggetto globale**.

Nel **browser**, l'oggetto globale è **window**



```
console.log(this); // window
```

Nota importante: in modalità **strict**, **this** nel contesto globale sarà **undefined**



```
'use strict';  
console.log(this); // undefined
```

Regole base di this

Dentro una funzione

Quando **this** è usato all'interno di una **funzione** che viene invocata nel **contesto globale**, this fa riferimento all'**oggetto globale** (sempre in modalità non-strict).

Nel **browser**, l'oggetto globale è **window**

```
function show() {  
  console.log(this);  
}  
show(); // window
```

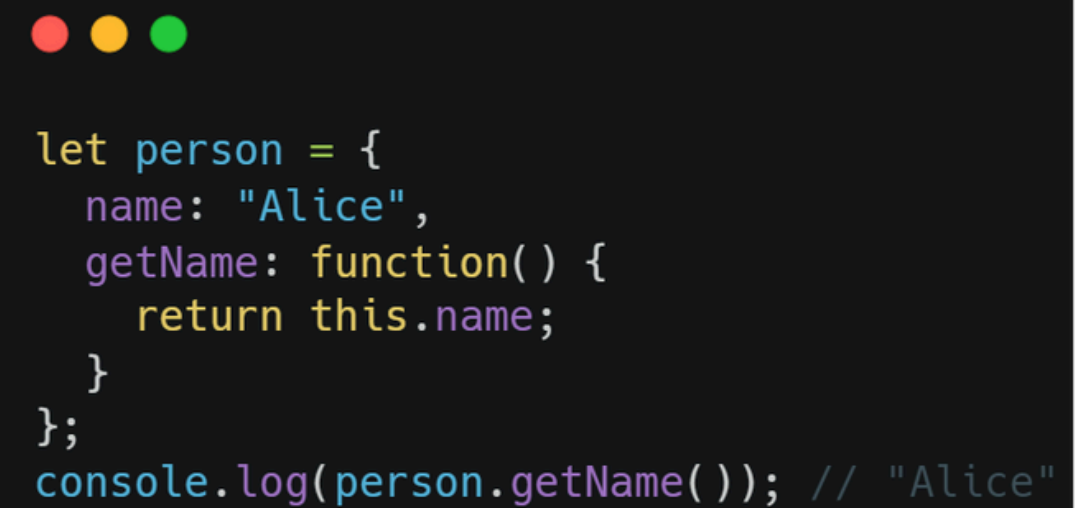
In **strict mode**, this dentro una funzione è **undefined**

```
'use strict';  
function show() {  
  console.log(this);  
}  
show(); // undefined
```

Regole base di this

Metodo di un oggetto

Quando una **funzione** è chiamata come **metodo di un oggetto**, **this** fa riferimento all'**oggetto** che contiene il metodo.



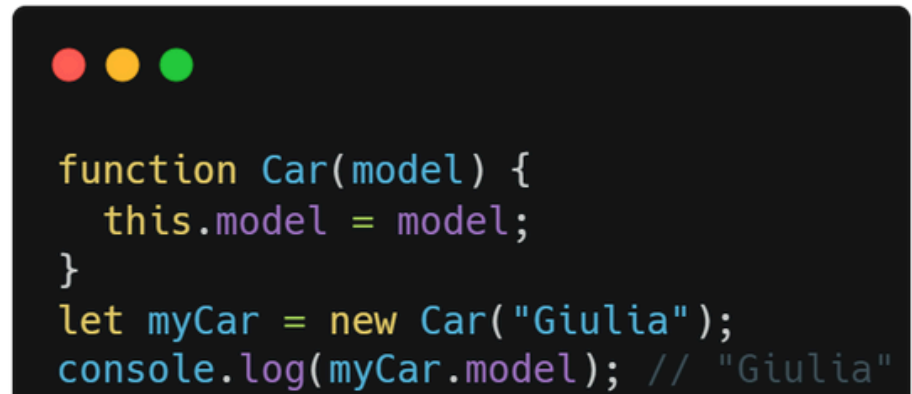
```
let person = {  
  name: "Alice",  
  getName: function() {  
    return this.name;  
  }  
};  
console.log(person.getName()); // "Alice"
```

Qui, **this** all'interno di `getName` si riferisce all'oggetto **person**.

Regole base di this

Funzione costruttrice

Quando una funzione viene utilizzata come **costruttrice** (usando **new**), **this** si riferisce all'**oggetto appena creato**.



```
function Car(model) {  
  this.model = model;  
}  
let myCar = new Car("Giulia");  
console.log(myCar.model); // "Giulia"
```

Qui, **this** all'interno della funzione costruttrice **Car** fa riferimento all'istanza di **Car** creata con **new**.

Problematiche comuni con this

Callback e this

Quando passi un metodo di un oggetto come **callback** o lo assigni a una **variabile**, **this** può comportarsi in **modo inatteso**. Il contesto di **this** potrebbe non essere più l'oggetto originale.

```
let user = {
  name: "John",
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
let greetFunc = user.greet;
greetFunc(); // "Hello, undefined" o in alcuni casi "Hello, window"
```

In questo caso, **this** non fa più riferimento a **user**, ma all'**oggetto globale** (o **undefined** in strict mode).

Problematiche comuni con this

Funzioni anonime e this

Funzioni anonime all'interno di metodi possono cambiare il contesto di **this**, portando a risultati inaspettati.

```
let user = {
  name: "Alice",
  showName: function() {
    setTimeout(function() {
      console.log(this.name);
    }, 1000);
  }
};
user.showName(); // undefined (o "window.name" in non-strict mode)
```

Qui, la **funzione anonima** passata a **setTimeout** viene eseguita nel **contesto globale**, quindi **this** non si riferisce più a user.

Soluzioni alle problematiche comuni

Variabile intermedia

Si può memorizzare il valore di **this** in una **variabile** (spesso chiamata **self** o **that**) e utilizzarla all'interno di funzioni annidate.

```
let user = {
  name: "Alice",
  showName: function() {
    let self = this; // memorizza il riferimento a this
    setTimeout(function() {
      console.log(self.name);
    }, 1000);
  }
};
user.showName(); // "Alice"
```

Qui, **self** memorizza il valore corretto di **this**, che si riferisce a **user**, e può essere utilizzato nella funzione di callback.

Soluzioni alle problematiche comuni

Arrow function

Le **arrow functions** (funzioni freccia) introducono un comportamento diverso per **this**. Esse non hanno il proprio this, ma **ereditano** il this dal contesto di esecuzione in cui sono definite.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains the following JavaScript code:

```
let user = {
  name: "Alice",
  showName: function() {
    setTimeout(() => {
      console.log(this.name);
    }, 1000);
  }
};
user.showName(); // "Alice"
```

Qui, la **funzione freccia** non crea un nuovo this, quindi **eredita** il **this** dell'oggetto user, risolvendo il problema.

Soluzioni alle problematiche comuni

Metodo .call()

Il metodo **.call()** ti consente di chiamare immediatamente una funzione con un determinato valore di **this** e un **elenco di argomenti separati da virgole**.

```
function saluta(prima, seconda) {  
  return `${prima} ${this.nome} ${seconda}`;  
}  
  
let persona = { nome: "Alice" };  
  
console.log(saluta.call(persona, "Ciao", "come stai?"));  
// "Ciao Alice come stai?"
```

In questo esempio, **.call()** imposta **this** su **persona**, rendendo **this.nome** disponibile all'interno della funzione.

Soluzioni alle problematiche comuni

Metodo .apply()

.apply() è simile a **.call()**, ma accetta un **array di argomenti** invece di un elenco separato da virgole.

```
function saluta(prima, seconda) {  
  return `${prima} ${this.nome} ${seconda}`;  
}  
  
let persona = { nome: "Alice" };  
  
console.log(saluta.apply(persona, ["Ciao", "come stai?"]));  
// "Ciao Alice come stai?"
```

La differenza con **.call()** sta nel modo in cui gli argomenti vengono passati.

Soluzioni alle problematiche comuni

Metodo .bind()

.bind() crea una **nuova funzione** con **this** legato a un determinato valore. A differenza di **.call()** e **.apply()**, **non esegue immediatamente la funzione**.

```
function saluta(prima, seconda) {  
  console.log(`${prima} ${this.nome} ${seconda}`);  
}  
  
let salutaAlice = saluta.bind(persona);  
  
salutaAlice("Ciao", "come stai?"); // "Ciao Alice come stai?"
```

Qui, **.bind()** crea una nuova funzione **salutaAlice**, che ha **this sempre legato a persona**, permettendoci di chiamare la funzione più tardi con la garanzia che **this** sarà quello desiderato.



Fonti:

- [Unschool](#): Understanding the "this" keyword in JavaScript
- [Dmitripavlutin](#): Gentle Explanation of "this" in JavaScript
- [Javascriptissexy](#): Understand JavaScript's "this" With Clarity, and Master It
- [Medium](#): JavaScript .call() .apply() and .bind() — explained to a total noob
- [Geeksforgeeks](#): What is the difference between call and apply in JavaScript?
- [MDN](#): this
- [MDN](#): Function.prototype.call()
- [MDN](#): Function.prototype.apply()
- [MDN](#): Function.prototype.bind()