



Lit

Google

Libreria per la creazione di web components

ELEONORA ROCCHI

- 2022 -

I componenti Lit ricevono input e memorizzano uno stato.

Le proprietà reattive sono proprietà che possono attivare il ciclo di aggiornamento reattivo quando vengono modificate, ri-eseguendo il rendering del componente e possono, facoltativamente, essere lette o scritte negli attributi.



```
class MyElement extends LitElement {  
  @property()  
  name: string;  
}
```


Lit gestisce

- ➔ **Aggiornamenti reattivi**: Lit genera una coppia getter/setter per ogni proprietà reattiva. Quando una proprietà reattiva cambia, il componente pianifica un aggiornamento.
- ➔ **Gestione degli attributi**: Per impostazione predefinita, Lit imposta un attributo osservato corrispondente alla proprietà e aggiorna la proprietà quando l'attributo cambia. I valori delle proprietà possono anche, facoltativamente, essere riflessi nell'attributo.
- ➔ **Proprietà di superclasse**: Lit applica automaticamente le opzioni di proprietà dichiarate da una superclasse. Non è necessario dichiarare nuovamente le proprietà a meno che non si desideri modificare le opzioni.
- ➔ **Aggiornamento dell'elemento**: Se un componente Lit viene definito dopo che l'elemento è già nel DOM, Lit gestisce la logica di aggiornamento, assicurando che qualsiasi proprietà impostata su un elemento prima che fosse aggiornato attivi gli effetti collaterali reattivi corretti quando l'elemento viene aggiornato.

Le proprietà pubbliche fanno parte dell'API pubblica del componente. In particolare le proprietà reattive pubbliche, dovrebbero essere trattate come input.

Il componente non dovrebbe modificare le proprie proprietà pubbliche, se non in risposta all'input dell'utente.

Lit supporta anche lo stato reattivo interno, che riferisce a proprietà reattive che non fanno parte dell'API del componente. Queste proprietà non hanno un attributo corrispondente e sono in genere contrassegnate come protette o private in TypeScript.



```
@state()  
private _counter = 0;
```


Il componente è in grado di manipolare il proprio stato reattivo interno.

In alcuni casi, lo stato reattivo interno può essere inizializzato da proprietà pubbliche, ad esempio se c'è una costosa trasformazione tra la proprietà visibile dall'utente e lo stato interno.


Come per le proprietà reattive pubbliche, l'aggiornamento dello stato reattivo interno attiva un ciclo di aggiornamento.

Le proprietà reattive pubbliche si dichiarano utilizzando il decoratore `@property`, eventualmente con una serie di opzioni.



```
class MyElement extends LitElement {  
  @property({type: String})  
  mode: string;  
  
  @property({attribute: false})  
  data = {};  
}
```


Oppure una proprietà statica:



```
class MyElement extends LitElement {  
  static properties = {  
    mode: {type: String},  
    data: {attribute: false},  
  };  
  
  constructor() {  
    super();  
    this.data = {};  
  }  
}
```


I campi di classe hanno un'interazione problematica con le proprietà reattive, perché definiti nell'istanza dell'elemento.

Le proprietà reattive sono definite come accessorie sul prototype dell'elemento.

Secondo le regole di JavaScript, una proprietà di istanza ha la precedenza e nasconde efficacemente una proprietà prototype.

Ciò significa che le funzioni di accesso alle proprietà reattive non funzionano quando vengono utilizzati i campi di classe.

Quando viene impostata una proprietà, l'elemento non si aggiorna.

In JavaScript non è necessario utilizzare i campi di classe quando si dichiarano proprietà reattive. Invece, le proprietà devono essere inizializzate nel costruttore dell'elemento.



```
constructor() {  
  super();  
  this.data = {};  
}
```


In TypeScript, si possono utilizzare i campi di classe per dichiarare le proprietà reattive purché utilizzi uno di questi modelli:

- ➡ Con impostazione `useDefineForClassFields` nel `tsconfig` su *false*.
- ➡ Aggiungendo la parola chiave *declare* sul campo e inserendo l'inizializzatore del campo nel costruttore.

Quando si compila JavaScript con Babel, è possibile utilizzare i campi di classe per dichiarare proprietà reattive purché si imposti *setPublicClassFields* su *true* nella configurazione dei presupposti del proprio `babelrc`.

L'oggetto opzioni può avere le seguenti proprietà:

- ➔ **attribute**: se la proprietà è associata a un attributo o un nome personalizzato per l'attributo associato. Predefinito: *true*. Se l'attributo è *false*, le opzioni *converter*, *reflect* e *type* vengono ignorate.
- ➔ **converter**: convertitore personalizzato per la conversione tra proprietà e attributi. Se non specificato, utilizzare il convertitore di attributi predefinito.
- ➔ **hasChanged**: funzione chiamata ogni volta che la proprietà è impostata per determinare se è stata modificata e dovrebbe attivare un aggiornamento. Se non specificato, LitElement utilizza un controllo rigoroso della disuguaglianza (*newValue !== oldValue*) per determinare se il valore della proprietà è cambiato.
- ➔ **noAccessor**: impostata su *true* per evitare di generare le funzioni di accesso alle proprietà predefinite. Questa opzione è raramente necessaria. Predefinito: *false*.

- ➔ **reflect**: se il valore della proprietà viene riflesso nell'attributo associato. Predefinito: *false*.
- ➔ **state**: da impostare su *true* per dichiarare la proprietà come stato reattivo interno. Lo stato reattivo interno attiva gli aggiornamenti come le proprietà reattive pubbliche, ma Lit non genera un attributo per esso e gli utenti non devono accedervi dall'esterno del componente. Equivalente all'utilizzo del decoratore `@state`. Predefinito: *false*.
- ➔ **type**: Quando si converte un attributo con valori di stringa in una proprietà, il convertitore di attributi predefinito di Lit analizzerà la stringa nel tipo specificato e viceversa quando riflette una proprietà in un attributo. Se il *converter* è impostato, questo campo viene passato al *converter*. Se il tipo non è specificato, il *converter* predefinito lo considera come tipo: `String`. Quando si utilizza TypeScript, dovrebbe generalmente corrispondere al tipo TypeScript dichiarato per il campo. Tuttavia, l'opzione *type* viene utilizzata dal runtime di Lit per la serializzazione/deserializzazione delle stringhe e non deve essere confusa con un meccanismo di controllo del tipo.

Omettere l'oggetto opzioni o specificare un oggetto opzioni vuoto equivale a specificare il valore predefinito per tutte le opzioni.