



Lit

Google

# Libreria per la creazione di web components

ELEONORA ROCCHI

- 2022 -



Lit si può facilmente aggiungere a progetti esistenti:  
i component Lit infatti funzionano con qualsiasi framework JavaScript,  
sistema di template server e CMS.



Un component Lit infatti può essere utilizzato direttamente nell'HTML, con le API DOM o nei template.

La maggior parte dei framework JavaScript ha un ottimo supporto per i Web component e Lit: è sufficiente importare la definizione dell'elemento ed utilizzare i nomi dei tag dell'elemento nei modelli.



# Come aggiungere Lit ad un progetto esistente



Per prima cosa, si installa il pacchetto Lit da riga di comando, con npm



```
> npm i lit
```



In qualsiasi cartella di progetto si può creare un nuovo elemento, utilizzando JSX



```
import {LitElement, html} from 'lit';
import {customElement} from 'lit/decorators.js';

@customElement('my-element')
class MyElement extends LitElement {
  render() {
    return html`
      <div>Hello from MyElement!</div>
    `;
  }
}
```



# Cos'è JSX

JavaScript Syntax Extension



JSX è un linguaggio utilizzato per la creazione di modelli.  
Si tratta di un'estensione della sintassi JavaScript.



# Cos'è un componente Lit



Un componente Lit è un pezzo riutilizzabile dell'interfaccia utente.

È una sorta di contenitore, con uno stato,  
che visualizza un'interfaccia utente in base al suo stato.

Può reagire all'input dell'utente, attivare eventi,  
qualsiasi ci si può aspettare da un componente dell'interfaccia utente.

Un componente Lit è un elemento HTML,  
quindi ha tutte le API degli elementi standard.



## Esempio:



```
import {LitElement, css, html} from 'lit';
import {customElement, property} from 'lit/decorators.js';

@customElement('simple-greeting')
export class SimpleGreeting extends LitElement {
  // Define scoped styles right with your component, in plain CSS
  static styles = css`
    :host {
      color: blue;
    }
  `;

  // Declare reactive properties
  @property()
  name?: string = 'World';

  // Render the UI as a function of component state
  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}
```



# Come definire un web component con Lit



Si crea una classe che estende LitElement e la si registra con il browser:



```
@customElement('simple-greeting')  
export class SimpleGreeting extends LitElement { /* ... */ }
```

In TS, il decoratore `@customElement` è un'abbreviazione per chiamare `customElements.define`, che registra una classe di elementi personalizzati con il browser e la associa a un nome di elemento (in questo caso, `simple-greeting`).

Altrimenti si può usare JS:



```
export class SimpleGreeting extends LitElement { /* ... */ }  
customElements.define('simple-greeting', SimpleGreeting);
```



Definendo un componente Lit, si definisce un elemento HTML personalizzato; si può usare il nuovo elemento come qualsiasi elemento HTML:



```
<simple-greeting name="Markup"></simple-greeting>
```



```
const greeting = document.createElement('simple-greeting');
```



**La classe base LitElement  
è una sotto-classe di HTMLElement,  
perciò un componente Lit eredita tutte le proprietà  
dei metodi standard di HTMLElement.**



**In particolare, LitElement eredita da ReactiveElement,  
che implementa proprietà reattive,  
e a sua volta eredita da HTMLElement.**



TypeScript deduce la classe di un elemento HTML restituito da alcune API DOM in base al nome del tag.

Sappiamo ad esempio che `document.createElement('img')` restituisce un'istanza `HTMLImageElement` con una proprietà `src` di tipo `string`.

Gli elementi personalizzati possono ottenere lo stesso trattamento aggiungendo la mappatura con `HTMLElementTagNameMap`.



```
@customElement('my-element')
export class MyElement extends LitElement {
  @property({type: Number})
  aNumber: number = 5;
  /* ... */
}

declare global {
  interface HTMLElementTagNameMap {
    "my-element": MyElement;
  }
}
```



In questo modo, il codice seguente esegue correttamente i controlli di tipo:



```
const myElement = document.createElement( 'my-element' );  
myElement.aNumber = 10;
```

Conviene aggiungere una voce `HTMLElementTagNameMap` a tutti gli elementi creati in TypeScript e pubblicare i tipi di `.d.ts` nel pacchetto npm.