

**Formazione  
specialistica  
per sviluppatore  
front-end**





# Hello!

## Alessandro Pasqualini

Full-stack developer, appassionato di  
programmazione e sviluppo software in generale, amo  
le serie tv e i gatti

1

# Introduzione oggetti JS



# Oggetti

Un oggetto è una collezione di dati e/o funzionalità correlati.

Nella programmazione ad oggetti si "uniscono" dati e funzioni (chiamate metodi).

In javascript sono molto utilizzati anche come "configurazione" di alcune librerie. Es. slick

```
$('.one-time').slick({  
    dots: true,  
    infinite: true,  
    speed: 300,  
    slidesToShow: 1,  
    adaptiveHeight: true  
});
```



# Proprietà degli oggetti

Gli oggetti sono variabili e quindi si istanziano (creano):

```
var persona = {};
```

Javascript assegna il tipo *object* alla variabile che contiene un oggetto.

Gli oggetti possono essere visti come un insieme di associazioni *chiave-valore* chiamate *proprietà*. Le proprietà sono delle variabili e quindi possono contenere tutto quello che una variabile può contenere.



# Proprietà degli oggetti

```
var persona = {  
    nome: 'John Smith',  
    anni: 33  
};  
  
console.log(persona);
```

```
▼ {nome: "John Smith", anni: 33} ⓘ  
  nome: "John Smith"  
  anni: 33  
► __proto__: Object
```

Naturalmente è possibile accedere ad ogni proprietà singolarmente usando la notazione `oggetto.proprieta`

```
var persona = {  
    nome: 'John Smith',  
    anni: 33  
};  
  
console.log(persona.nome);
```

John Smith



# Metodi degli oggetti

Gli oggetti sono collezioni di dati e funzionalità. I dati sono rappresentati dalle proprietà, mentre le funzionalità dai metodi.

I metodi sono delle funzioni definite all'interno dell'oggetto e possono usare *le proprietà del loro oggetto*.

```
var persona = {  
    nome: 'John Smith',  
    saluta: function () {  
        console.log('Ciao');  
    }  
};
```



# Metodi degli oggetti

Per usare un metodo è necessario invocarlo (chiamarlo).

```
var persona = {  
    nome: 'John Smith',  
    saluta: function () {  
        console.log('Ciao');  
    }  
};  
  
persona.saluta();
```

---

Ciao

---



# Metodi degli oggetti

I metodi possono accedere anche alle proprietà dell'oggetto in cui sono definite.

```
var persona = {  
    nome: 'John Smith',  
    saluta: function () {  
        console.log(this.nome);  
    }  
};  
  
persona.saluta();
```

---

John Smith

---



# Metodi degli oggetti

La parola chiave **this** può essere usata all'interno dei metodi per indicare l'oggetto in cui sono definiti.

Viene usata se il metodo deve accedere ad una proprietà dell'oggetto.

```
var persona = {  
    nome: 'John Smith',  
    saluta: function () {  
        console.log(this.nome);  
    }  
};  
  
persona.saluta();
```

---

John Smith

---

1

# JSON (JavaScript Object Notation)



# JSON

JSON è un formato per il trasporto dei dati e deriva dalla notazione per gli oggetti di Javascript.

JSON è indipendente dal linguaggio usato. Si può usare anche con altri linguaggi di programmazione, ad esempio PHP, Java, etc

JSON sta sostituendo XML (un formato "simile" ad HTML) per il trasporto dei dati perchè è molto leggero (poca formattazione) ed è molto facile da leggere anche per gli umani.



# XML VS JSON

```
{  
  "id": 123,  
  "title": "Object Thinking",  
  "author": "David West",  
  "published": {  
    "by": "Microsoft Press",  
    "year": 2004  
  }  
}
```

JSON

```
<?xml version="1.0"?>  
<book id="123">  
  <title>Object Thinking</title>  
  <author>David West</author>  
  <published>  
    <by>Microsoft Press</by>  
    <year>2004</year>  
  </published>  
</book>
```

XML



# La sintassi

JSON deriva dalla sintassi degli oggetti di Javascript e quindi è molto simile alla loro dichiarazione.

```
{  
    "prop1": valore1,  
    "prop2": valore2,  
    ...  
}
```

JSON supporta stringhe, numeri, array, oggetti, boolean, null (è un tipo speciale)

2

# Ajax (Asynchronous Javascript And Xml)



# Ajax

Ajax è la tecnologia più usata per realizzare applicazioni interattive. Il concetto che sta alla base è quello di scambiare dati con il browser senza dover ricaricare la pagina.

jQuery ci permette di realizzare delle chiamate AJAX con pochissimo sforzo.

Le chiamate AJAX sono chiamate HTTP classiche (GET, POST, etc).

AJAX originariamente usava XML come formato di trasporto dei dati ma è stato quasi completamente sostituito da JSON (e da altri formati)



# \$load

\$load è un metodo di jQuery (\$ di jQuery è un oggetto) che ci permette di caricare del contenuto (principalmente HTML) in modo asincrono.

```
<button id="clICCami">ClICCami</button>
<div id="contenuto"></div>
<script>
    $('#clICCami').click(function () {
        $('#contenuto').load('snippet.html', function () {
            console.log('Contenuto caricato');
        });
    }
</script>
```



# \$load

Cliccami

Cliccami





# \$load

Il file snippet.html

```

```



# \$.load

Name	Status	Type
corso	200	document
jquery.min.js	200	script

XHR significa XMLHttpRequest.

XMLHttpRequest è l'API (Application Program Interface) messa a disposizione dal browser per fare richieste asincrone

Name	Status	Type
corso	200	document
jquery.min.js	200	script
snippet.html	200	xhr
cat.png	200	png



# \$.get

\$.get è un metodo di jQuery che ci permette di effettuare una richiesta HTTP GET per prelevare del contenuto in maniera asincrona.

```
$(document).ready(function () {  
    $.get('users.json', function (usernames) {  
        console.log(usernames);  
    });  
});
```



# \$.get

```
▼ (2) [{...}, {...}] ⓘ  
► 0: {username: "ilmiousername1", signupDate: 1578820209000}  
► 1: {username: "ilmiousername2", signupDate: 1578820209000}  
  length: 2  
► __proto__: Array(0)
```

```
[{  
  "username": "ilmiousername1",  
  "signupDate": 1578820209000  
, {  
  "username": "ilmiousername2",  
  "signupDate": 1578820209000  
}]
```

Il file  
*users.json*



# **\$(document).ready(...)**

\$(document).ready(...) ci permette di registrare un callback (una funzione) da eseguire quando il browser emettere l'evento "pagina pronta".

Questo ci permette di ritardare l'esecuzione del codice fino a che la pagina non è pronta.

```
$(document).ready(function () {  
    console.log('La pagina è pronta');  
});  
console.log('Scrivi subito');
```

Scrivi subito
La pagina è pronta



# **\$(window).on('load', ...)**

`$(document).load(...)` ci permette di registrare un callback (una funzione) da eseguire quando il browser emettere l'evento "pagina completamente caricata" comprensiva di immagini, iframe etc.

```
$(window).on('load', function () {  
    console.log('La pagina è caricata');  
});  
$(document).ready(function () {  
    console.log('La pagina è pronta');  
});  
console.log('Scrivi subito');
```

Scrivi subito

La pagina è pronta

La pagina è caricata

.



# **\$(window).on('load', ...)**

Attenzione che **load** funziona solo su **window** e non **document**.

```
$(document).on('load', function () {  
    console.log('La pagina è caricata');  
});
```



```
$(window).on('load', function () {  
    console.log('La pagina è caricata');  
});
```

La pagina è caricata





# \$.**post**

\$.**post** è un metodo di jQuery che ci permette di effettuare una richiesta HTTP POST per inviare del contenuto al server in modo asincrono.

\$.**post** accetta un oggetto Javascript contenente i dati da inviare, quindi il nostro compito è crearlo inserendo i dati da comunicare al server.

\$.**post** permette anche di impostare un il tipo dei dati che il server ci ritornerà (ad esempio json). Se non è impostato viene considerato testo e quindi l'eventuale json NON è automaticamente trasformato in un oggetto Javascript.



# \$ .post

```
$.post(  
    'url_della_pagina',  
    { ... },  
    function (risposta) {  
        ...  
    },  
    'tipo_dei_dati_in_risposta_dal_server'  
);
```



# \$.ajax

jQuery mette a disposizione dei programmati anche un metodo chiamato `$.ajax` che permette di avere un controllo maggiore sulla richiesta asincrona.

`$.load`, `$.get` e `$.post` sono solamente delle "scorciatoie" per `$.ajax`.

Nella documentazione di jQuery è possibile trovare tutti i parametri accettati da questi metodi per modificarne il comportamento. Ad esempio è possibile inserire nella richiesta HTTP headers aggiuntivi.



# Il problema degli event handler

jQuery ci permette di registrare delle funzioni da eseguire quando si verifica un certo evento. es. click

```
$('#id').click(function () { ... });
```

Questi event handler sono però registrati al caricamento della pagina sul contenuto attuale. Se carichiamo del nuovo contenuto HTML gli event handler NON vengono registrati nel il nuovo codice.



# Il problema degli event handler

```
<button class="clICCami">ClICCami 1</button>
<script>
    $('.clICCami').click(function () {
        alert('ok');
    });
</script>
```

Se ora aggiungiamo un nuovo bottone attraverso del codice JS (magari caricandolo con \$.load)...



# Il problema degli event handler

```
<button class="cliccammi">Cliccammi 1</button>
```

```
<button class="cliccammi">Nuovo bottone</button>
```

```
<script>
  $('.cliccammi').click(function () {
    alert('ok');
  });
</script>
```

L'event handler non funziona per il nuovo bottone



# Il problema degli event handler

La soluzione è registrare l'event handler a livello di document

```
$('.cliccammi').click(function () {  
    alert('ok');  
});
```

```
$(document).click('.cliccammi', function () {  
    alert('ok');  
});
```



# CORS

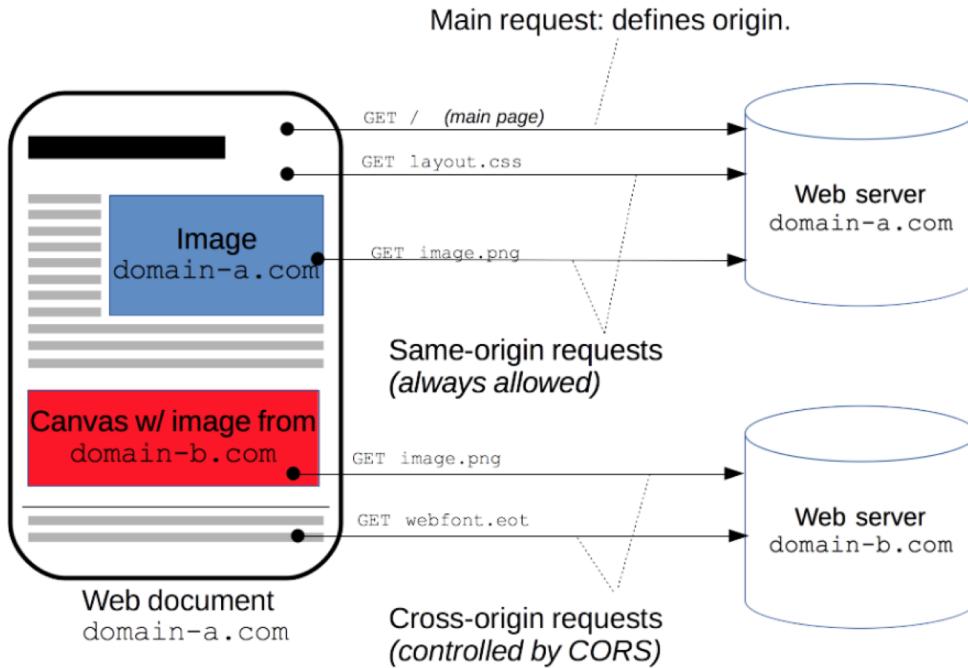
CORS (Cross Origin Resource Sharing) è un meccanismo usato per indicare al browser che l'applicazione in esecuzione su una specifica origine (il dominio) ha l'autorizzazione di accedere a risorse su un'altra origine (altro dominio).

Per ragioni di sicurezza i browser limitano l'utilizzo di CORS per proteggere l'utente da situazioni pericolose.

Tutte le richieste CORS che provengono da script (Javascript) vengono sottoposte al controllo del browser prima di consentire la richiesta.



# CORS





# CORS

Riprendendo l'esempio di `$.load` ma caricandolo all'esterno del web server

```
<button id="clICCami">ClICCami</button>
<div id="contenuto"></div>
<script>
    $('#clICCami').click(function () {
        $('#contenuto').load('snippet.html', function () {
            console.log('Contenuto caricato');
        });
    }
</script>
```



# CORS

Cliccami

✖ ► Access to XMLHttpRequest at 'file:///Users/alessandro/Desktop/Corso/Slides/2020\_01\_13/test/snippet.html' from origin 'null' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: http, data, chrome, chrome-extension, https.

>



# CORS

Riprendendo l'esempio di `$.load` ma caricandolo all'esterno del web server

```
<button id="clICCami">ClICCami</button>
<div id="contenuto"></div>
<script>
    $('#clICCami').click(function () {
        $('#contenuto').load('http://localhost/snippet.html', function () {
            console.log('Contenuto caricato');
        });
    }
</script>
```



# CORS

Cliccami

- ✖ Access to XMLHttpRequest at '<http://localhost/snippet.html>' sop.html:1 from origin '<http://corso>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

3

**Esempio: la chat**



# Thanks!

## Any questions?

Per qualsiasi domanda contattatemi:  
[alessandro.pasqualini.1105@gmail.com](mailto:alessandro.pasqualini.1105@gmail.com)