

Web Developer

Programmazione - Javascript e Typescript

Docente: Shadi Lahham

Modules and build tools

BUILDING: modifica del codice Building and bundling code

Shadi Lahham - Web development

SONO STATI INTRODOTTI IN ECMAScript nel 2015 --> ES6

Modules

I moduli in JavaScript sono blocchi autonomi di codice che consentono di organizzare, separare e riutilizzare le funzionalità del programma in modo più efficace.

Ogni modulo è un file che contiene funzioni, oggetti, classi o variabili, e può esportare una o più parti del suo contenuto per essere utilizzate in altri file.

Modules

- Import from other files [Puoi importare variabili, funzioni, oggetti da altri file usando IMPORT.](#)
 - Variables, functions, objects, etc
- **Modern browsers only**
 - [Browser compatibility](#)
- Don't work with local files
 - file:///url
- Need an HTTP server
 - [http-server](#), [NginX](#), [Apache](#), [Wamp](#), [XAMPP](#)
 - [Live Server - VSCode ext](#)
 - [The 8 Best Open Source Web Servers](#)

Modules

Modules

- A self-contained unit of code
- Encapsulates specific functionality for better organization and reusability
- Facilitates dependency management
- Supports code reuse and collaboration on projects
- Promotes clean and manageable code through separation of concerns

Module vs regular JavaScript files

- Variables and functions are scoped to the module by default for better encapsulation and avoiding global scope pollution
- Modules have their own top-level scope to avoid conflicts with other modules
- `import` and `export` are used to define module dependencies

[JavaScript modules](#)

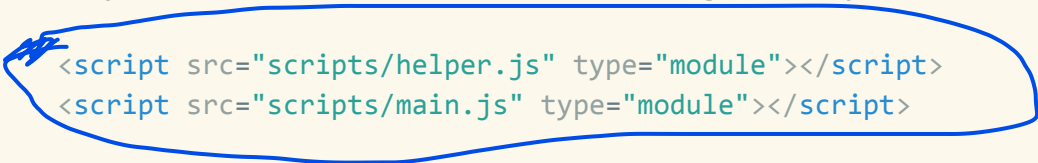
Modules

Loading

When a browser encounters a module script with `type="module"` or `.mjs`, it loads the module file asynchronously while parsing and rendering the HTML. Module scripts are *deferred* by default

Execution

When the parser finishes, modules are executed in the order they are encountered in the HTML, ensuring they run after the entire document, along with any dependencies, has been fully parsed



```
<script src="scripts/helper.js" type="module"></script>  
<script src="scripts/main.js" type="module"></script>
```

usare questo sempre!!

Alternative, but not recommended by [MDN](#)

```
<script src="scripts/helper.mjs"></script>  
<script src="scripts/main.mjs"></script>
```

Use the *async* attribute if you want the browser to execute the module immediately upon loading

```
<script type="module" src="module.js" async></script>
```

DEFAULT --> in una funzione serve per indicare che quella funzione è l'esportazione predefinita (o principale) di un modulo.

Export & import

Questo significa che, quando importi il modulo in un altro file, non devi usare un nome specifico per l'importazione;

```
<script src="scripts/helper.js" type="module"></script>
<script src="scripts/main.js" type="module"></script>
```

```
// helper.js -- exports a default function
```

```
export default function(msg) {
  return `I am a helper function. You passed "${msg}"`;
}
```

```
// main.js
```

```
import helper from './helper.js';
```

```
const msg = helper('external help');
```

```
console.log(msg);
```

importiamo la funzione esportata (chiamata helper in questo file) from il file di esportazione

Multiple exports

```
// helper.js
```

```
export default (msg) => `Echo "${msg}"`; 01. Esportazione di default --> una funzione anonima che prende un messaggio (msg) e lo restituisce  
const user = { name: 'james' };         con il prefisso "Echo". Questa è l'esportazione principale di helper.js.  
const double = (x) => x * 2;             02. Esportazioni nominate -->  
export { user, double };                - user: un oggetto con una proprietà name impostata su "james".  
                                         - double: una funzione che moltiplica un numero per 2.
```

```
// main.js
```

```
import echo, { user, double } from './helper.js'; // named imports must match  
console.log(echo('external help'));  
console.log(user.name);  
console.log(double(5));
```


Renaming imports

// helper.js

```
export default (msg) => `Echo "${msg}"`;
const user = { name: 'james' };
const double = (x) => x * 2;
export { user, double };
```

// main.js

```
import anyName from './helper.js'; // imports the default export as anyName
console.log(anyName('external help'));
```

// alternative main.js - renaming named imports

```
import anyName, { user as someone, double as timesTwo } from './helper.js';
console.log(anyName('external help'));
console.log(someone.name);
console.log(timesTwo(5));
```

Renaming imports

```
// helper.js
```

```
export default (msg) => `Echo "${msg}"`;
const user = { name: 'james' };
const double = (x) => x * 2;
export { user, double };
```

```
// alternative main.js - renaming default
```

```
import { default as repeat, user as someone, double as timesTwo } from './helper.js';
console.log(repeat('external help'));
```

```
// alternative main.js - import all
```

```
import * as everything from './helper.js'; // renaming named imports
console.log(everything.default('external help'));
console.log(everything.user.name);
console.log(everything.double(5));
```

Best practice

```
// main.js  
import helper from './helper.js'; // import just one entity, name it meaningfully  
console.log(helper.name);
```

```
// helper.js  
const api = {  
  name: 'something'  
  // ...  
};  
export default api; // export only one entity and export it as default
```

IIFE

Una funzione che viene eseguita immediatamente dopo essere stata dichiarata. È utile per creare scope separati, ma non è più necessaria con le moderne pratiche JavaScript.

è come se fosse una funzione usa e getta

What is an IIFE

Immediately invoked function expression

- The old way to create a scope for variables declared using var other than function scope
- Use in the past to create 'modules' of separated responsibilities
- Not required in modern Javascript
- You might encounter them in old code or isolated situations

IIFE example

sintassi per creare le IIFE

// declare and immediately call a function

```
(function () {  
  var message = 'Hello';  
  console.log(message); // message exists in this scope  
})();
```

```
console.log(message); // message is not defined
```

What is an IIFE

[IIFE - MDN](#)

[Immediately invoked function expression](#)

[Why it's Time to Stop Using JavaScript IIFEs](#)

[The many ways to write an IIFE](#)

[I Love My IIFE](#)

[Immediately-Invoked Function Expression \(IIFE\)](#)

Minification

per rendere i file il + leggeri e minimi possibili

What is minification

- removing unnecessary or redundant data
- without affecting how the resource is processed by the browser
 - removing code comments
 - removing unnecessary spaces and formatting
 - removing unused code
 - using shorter variable and function names
- Possible to minify HTML, CSS and Javascript

Javascript minification

main.js

```
let greeter = (name, place) => `Mister ${name}  
of${place}`;
```

// function with a side effect

```
let nameLogger = (name, place) => {  
  let newName = `Mister ${name} of${place}`;  
  console.log(newName);  
  return newName;  
};
```

main.min.js

```
let greeter=(e,r)=>`Mister ${e}  
of${r}`,nameLogger=(e,r)=>{let o=`Mister ${e}  
of${r}`;return console.log(o),o};
```

[UglifyJS 3: Online JavaScript minifier](#)

CSS minification

style.css

```
.geo-image {  
  margin: 0 auto;  
  width: 100%;  
  position: relative;  
}  
.geo-image img {  
  width: 100%;  
  display: block;  
}  
  
/* image caption */  
.geo-image figcaption {  
  background-color: orange;  
  position: absolute;  
  bottom: 8px;  
}
```

style.min.css

```
.geo-image{margin:0  
auto;width:100%;position:relative}.geo-image  
img{width:100%;display:block}.geo-image  
figcaption{background-color:orange;position:absol  
ute;left:-4px;bottom:8px}
```

[Minify CSS and JS](#)

Content Delivery Network

Una CDN (Content Delivery Network) è un insieme di server distribuiti in varie località geografiche che lavorano insieme per fornire contenuti agli utenti in modo più efficiente.

L'obiettivo principale di una CDN è ridurre la latenza e migliorare la velocità di caricamento dei contenuti per gli utenti finali, avvicinando i contenuti alla loro posizione fisica.

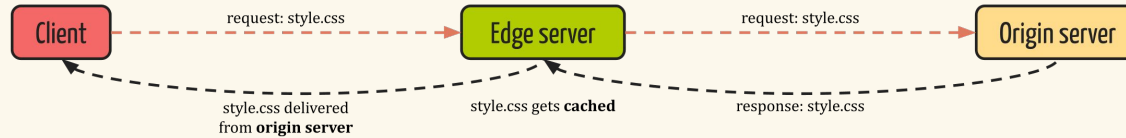
CDN - Content Delivery Network

- A large distributed system of servers deployed in multiple data centers across the Internet
- Serve content to end-users with high availability and high performance
- **CDNs serve a large fraction of the Internet content today**
 - web objects: text, graphics and scripts
 - downloadable objects: media files, software, documents
 - applications: e-commerce, portals
 - live streaming media
 - on-demand streaming media
 - social networks

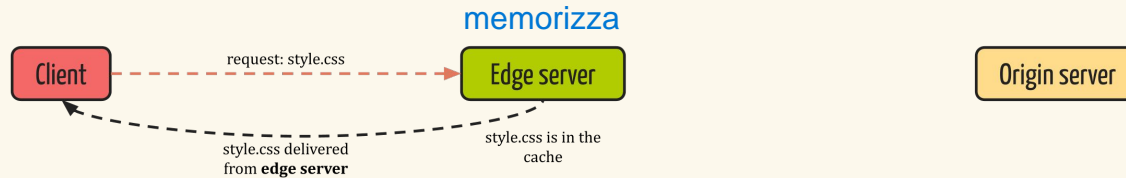
How CDNs work

Content Delivery Network

First request



Subsequent requests



How CDNs work

Content Delivery Network



How CDNs work

Network without a CDN



Using a CDN

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

<script crossorigin src="https://unpkg.com/react@17/umd/react.production.min.js"></script>
<script crossorigin src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.21/lodash.min.js"></script>
```

Some known CDNs

[Cloudflare CDN](#)

[Google cloud CDN](#)

[Amazon CloudFront CDN](#)

[UNPKG](#)

CDN package search

[cdnjs](#)

Build tools

Tools

Required to bundle code and also make modern code on older browsers

- [NodeJS](#)
 - package manager [npm](#)
 - installs packages
- [Babel](#)
 - compiler/transpiler [da webpack](#)
 - convert modern Javascript to versions compatible with older browsers - [documentation](#)
- [Webpack](#)
 - bundler
 - parses file imports to create bundles - [documentation](#)

Babel

Babel translates modern Javascript to ES5 that older browsers understand

Technically Babel is a JS [transpiler](#) which can

- Transform syntax
- Polyfill missing features
- Perform source code transformations

[Try Babel](#) with your own code

Also, use [Browserslist](#) for understanding targets

utilizzato per la FASE DI SVILUPPO, sono coinvolti vari strumenti per migliorare l'efficienza del processo di creazione e della sua qualità, MA SONO STRUMENTI IDONEI ALLA FASE SOLO DI SVILUPPO!!!, come:

Webpack

- SASS al posto di CSS
- utilizzo di JS moderno
- utilizzo di TYPESCRIPT

Webpack is a bundler that packs many JS module files into one bundle file

- Can produce one or a few bundles, e.g. bundle.js
- Supports many module systems
- Can be used for other types of files CSS, Images, JSON, SASS, etc
- Can minify and uglify Javascript code
- Managed by a configuration file called webpack.config.js and a CLI

sono tutti strumenti che NON VANNO D'ACCORDO CON IL BROWESER (produzione), ci dobbiamo munire di compilatori per trasformare i vari preprocessori (SASS JS moderno...) in qualcosa che è comprensibile per il browser

queste operazioni possiamo AUTOMATIZZARE GRAZIE AI MODULE BLANDER (WEBPACK)

1. fase SVILUPPO src/

2. fase PRODUZIONE (risultato finale dopo una serie di ottimizzazioni del codice sorgente) dist/

Setup

Install [NodeJS](#) if it is not already installed

Create the following project folder and files:

```
project
├── src
│   ├── index.html
│   ├── scripts
│   │   ├── main.js
│   │   └── helper.js
│   └── styles
│       └── main.css
├── static
│   └── images
├── package.json
└── webpack.config.js
```

eccetto lui creato da npm

configurazione già creata da vecchi file

tutto vuoto inizialmente

Modules to install

-D --> Il comando -D in npm (Node Package Manager) è un'opzione che si utilizza per installare un pacchetto come dipendenza di sviluppo (development dependency) in un progetto Node.js. Quando un pacchetto viene installato con questa opzione, viene registrato nella sezione devDependencies del file package.json del progetto.

// run the following commands to install the required modules

```
npm i -D babel-loader @babel/core @babel/preset-env
npm i -D webpack webpack-dev-server webpack-cli
npm i -D html-webpack-plugin
npm i -D css-loader mini-css-extract-plugin
npm i core-js@3
```

index.html

!!!! non vengono inclusi al document i vari file di style e scripts !!!!!

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title>JS Webpack</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>

<body>
  <!-- body content -->
  created with Webpack
</body>

</html>
```


main.css i file CSS / SCSS /SASS... vanno inclusi nell'entrypoint.js !!!!!!

```
body {  
  background: no-repeat url("../static/images/robot.jpg");  
  background-color: orangered;  
}
```

Javascript files

```
// main.js
import './styles/main.css';
import helper from './helper.js';

console.log(`I can run modern Javascript on older browsers`);
console.log(`message from helper: ${helper.msg}`);
const test = [1, 2, 3].includes(1);
console.log('test = ', test);
```

```
// helper.js
const api = {
  msg: 'I can use modules'
};
export default api;
```

package.json with webpack 5 dependencies

```
{
  "scripts": {
    "watch": "webpack --watch",
    "start": "webpack serve --open",
    "build": "webpack"
  },
  "devDependencies": {
    "@babel/core": "^7.25.2",
    "@babel/preset-env": "^7.25.3",
    "babel-loader": "^9.1.3",
    "css-loader": "^7.1.2",
    "html-webpack-plugin": "^5.6.0",
    "mini-css-extract-plugin": "^2.9.0",
    "webpack": "^5.93.0",
    "webpack-cli": "^5.1.4",
    "webpack-dev-server": "^5.0.4"
  },
  "dependencies": {
    "core-js": "^3.38.0"
  }
}
```

webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {
  mode: 'production',
  entry: './src/scripts/main.js',
  output: { filename: '[name].bundle.js', path: path.resolve(__dirname, 'dist') },
  devServer: {
    static: {
      directory: path.join(__dirname, 'static'),
      publicPath: '/static',
      serveIndex: true // http://localhost:8080/static/ is exposed; use only for learning
    }
  },
  plugins: [new HtmlWebpackPlugin({ template: './src/index.html' }), new MiniCssExtractPlugin()],
```

webpack.config.js -- continued

```
module: {
  rules: [
    { test: /\.css$/i, use: [MiniCssExtractPlugin.loader, 'css-loader'] },
    { test: /\.m?js$/, babel opera su file di tipo JAVASCRIPT (js), e di MJS è un modo di dire che è un file di tipo JAVASCRIPT(m)
      exclude: /(node_modules|bower_components)/,
      use: {
        loader: 'babel-loader', utilizza babel
        options: {
          presets: [[
            '@babel/preset-env',
            { targets: { edge: '127', firefox: '128', chrome: '127', safari: '17.5', ie: '11' },
              // targets: '> 0.25%, not dead',
              useBuiltIns: 'usage',
              corejs: '3.21.1' }
          ]]
        }
      }
  ]
};
```

Running

install dependencies (wait for installation to finish):

```
npm install
```

run webpack dev server:

```
npm start
```

build project bundle:

```
npm run build
```

Testing

Try your bundled code on various desktop and mobile browsers

Additionally, check websites like [Statcounter](#) and [Similarweb](#) to compare browser marketshare

Babel & Webpack

detailed configuration

Babel configuration options

The `targets` option specifies the environments your project supports and can be:

- A `browserslist`-compatible query
 - `targets: "> 0.25%, not dead"`
- An object listing minimum versions to support
 - `targets: { edge: '127', firefox: '128', chrome: '127', safari: '17.5', ie: '11' }`

If the `targets` option is not included in Babel, it will look for a `.browserslistrc` file in your project that defines the environments to support and use that configuration instead

If neither `targets` nor a `.browserslistrc` file is provided, Babel defaults to supporting a wide range of environments, leading to more polyfills and transformations, which can increase the bundle size

Babel configuration options

The `useBuiltIns` option controls how polyfills are included

- **entry** - manually import ``core-js`` at the entry point
 - Babel includes all necessary polyfills based on the **targets**
- **usage** - automatically adds imports for the specific polyfills needed in each file based on the code and **targets**, reducing bundle size

The `corejs` option specifies the version of core-js to use for polyfilling

Necessary when using the `useBuiltIns` option to automatically include only the polyfills needed based on the **targets** configuration

Using package.json

In package.json, you can define scripts for different environments

- **dev** - "webpack serve --mode development"
This script runs Webpack in development mode, which is ideal for building and testing during development. It enables features like detailed error messages, live reloading, and unminified output
- **build** - "webpack --mode production"
This script runs Webpack in production mode, optimizing the code for deployment. It minifies the output, removes unnecessary code, and generally makes the build more efficient for production

```
{
  "scripts": {
    "dev": "webpack serve --mode development",
    "build": "webpack --mode production"
  }
}
```

Using separate configuration files

Another way to manage different environments is by creating separate Webpack configuration files:

- **webpack.config.dev.js**
This file would contain settings optimized for development. You can specify the mode and other settings like source maps, dev server configurations, etc
- **webpack.config.prod.js**
This file would contain settings optimized for production, focusing on performance, minification, and other optimizations
- The common configurations are stored in `webpack.config.common.js`, and both `webpack.config.dev.js` and `webpack.config.prod.js` will extend this common configuration

```
{  
  "scripts": {  
    "dev": "webpack --config webpack.config.dev.js",  
    "build": "webpack --config webpack.config.prod.js"  
  }  
}
```

webpack.config.common.js

```
// webpack.config.common.js
const path = require('path');

module.exports = {
  entry: './src/index.js', // Common entry point
  output: {
    path: path.resolve(__dirname, 'dist')
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader' // common loader configuration
        }
      }
    ]
  }
};
```

webpack.config.dev.js

```
// webpack.config.dev.js
const { merge } = require('webpack-merge');
const common = require('./webpack.config.common.js');

module.exports = merge(common, {
  mode: 'development',
  devtool: 'inline-source-map',
  devServer: {
    static: './dist',
    hot: true // enable Hot Module Replacement
  },
  output: {
    filename: '[name].bundle.js' // common filename for development
  }
});
```

webpack.config.prod.js

[contenthash]

per gestire risolvere il problema della cache (versioni diverse di files)

cache -> versione vecchia

server -> versione nuova

```
// webpack.config.prod.js
const { merge } = require('webpack-merge');
const common = require('./webpack.config.common.js');

module.exports = merge(common, {
  mode: 'production',
  devtool: 'source-map', // source maps for production debugging
  output: {
    filename: '[name].[contenthash].bundle.js', // contenthash is used for cache busting in production
    clean: true // automatically clean the output directory before each build
  },
  optimization: {
    minimize: true // minimize the output
  }
});
```

Webpack configuration

[Concepts | webpack](#)

[Mode | webpack](#)

[DevServer | webpack](#)

[DevServer.static | webpack](#)

[webpack-dev-server - npm](#)

[Output | webpack](#)

[Development - watch mode | webpack](#)

[Set Up Webpack 5 To Work With Static Files | Older but useful guide](#)

Your turn

1.To CDN or not to CDN

Find sources similar to [this article](#) to understand the pros and cons of a CDN

- Write down as many pros and cons as you can think of
 - Explain why you think they are relevant
- Describe 2 scenarios where you think a CDN is required and 2 where it's not
 - Your examples should be realistic and should emphasize the pros or cons

Summarize your findings in a properly named markdown file

- [Markdown Guide](#)
- [Online Markdown Editor - Dillinger](#)

2. Webpack friendly

- Implement some of exercises of the previous units as a webpack project
- The aims are
 - to rewrite the same exercises with modern JS syntax
 - to use webpack, and polyfills if necessary, to make the code compatible with the largest number of browsers
- Document any important configuration or code changes in readme.md
- Test the projects with the largest number of browsers you can

Bonus

3. Bundler showcase

Explore [Parcel](#) or another [bundler of your choice](#)

- Create a small project with HTML, CSS, and JavaScript files
- Use the bundler to build and serve your project
- Ensure the output is optimized and the project runs correctly on various browsers
- Include a README.md with a brief explanation of how the bundler handled your files and any notable features or issues

[3 better alternatives for building your javascript](#)

[Top 5 alternatives to webpack](#)

4. Stylish bundling challenge

Use the following resources:

- [style-loader](#), [sass-loader](#), [Asset Modules Guide](#)

Create a small project with HTML, SCSS, JavaScript files, and images

- Use Webpack to:
 1. Build SCSS files into a single CSS bundle
 2. Handle various image types with automatic decision between inlining and emitting based on file size (e.g., 50KB)
- Include a README.md file with a brief explanation of your Webpack configuration and how it handles SCSS and images

References

[Import](#)

[Export](#)

[Global variables and JavaScript modules](#)

[Global Variables in JavaScript](#)

References

webpack

[DevServer](#)

[Development](#)

[Output](#)

webpack additions

[HtmlWebpackPlugin](#)

[css-loader](#)

[sass-loader](#)

References

[Webpack 5 : Guide for beginners](#)

[Setting up the Webpack Dev Server](#)

[How to Webpack 5 - Setup Tutorial](#)

[Set up Webpack 5 for Basic Javascript Projects](#)

[Setting Up Webpack for JavaScript, TypeScript and using Webpack Server](#)