Tecnologie e applicazioni web

Electron

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2021/2022

Electron

Electron is an open-source library developed in 2014 by GitHub to create cross-platform desktop applications using web technologies:

- Node.js JavaScript runtime for app backend
- Chromium browser (HTML/CSS) for user interface rendering

Electron creates a single executable package for Windows, Linux, and OSX

Electron

Lets you develop desktop applications in pure JavaScript, providing APIs to interface with the underlying operating system.

Note: Electron does not provide a JavaScript binding to native GUI APIs. The user interface is realized by visualizing a “minimal” JavaScript-controllable Chromium browser.

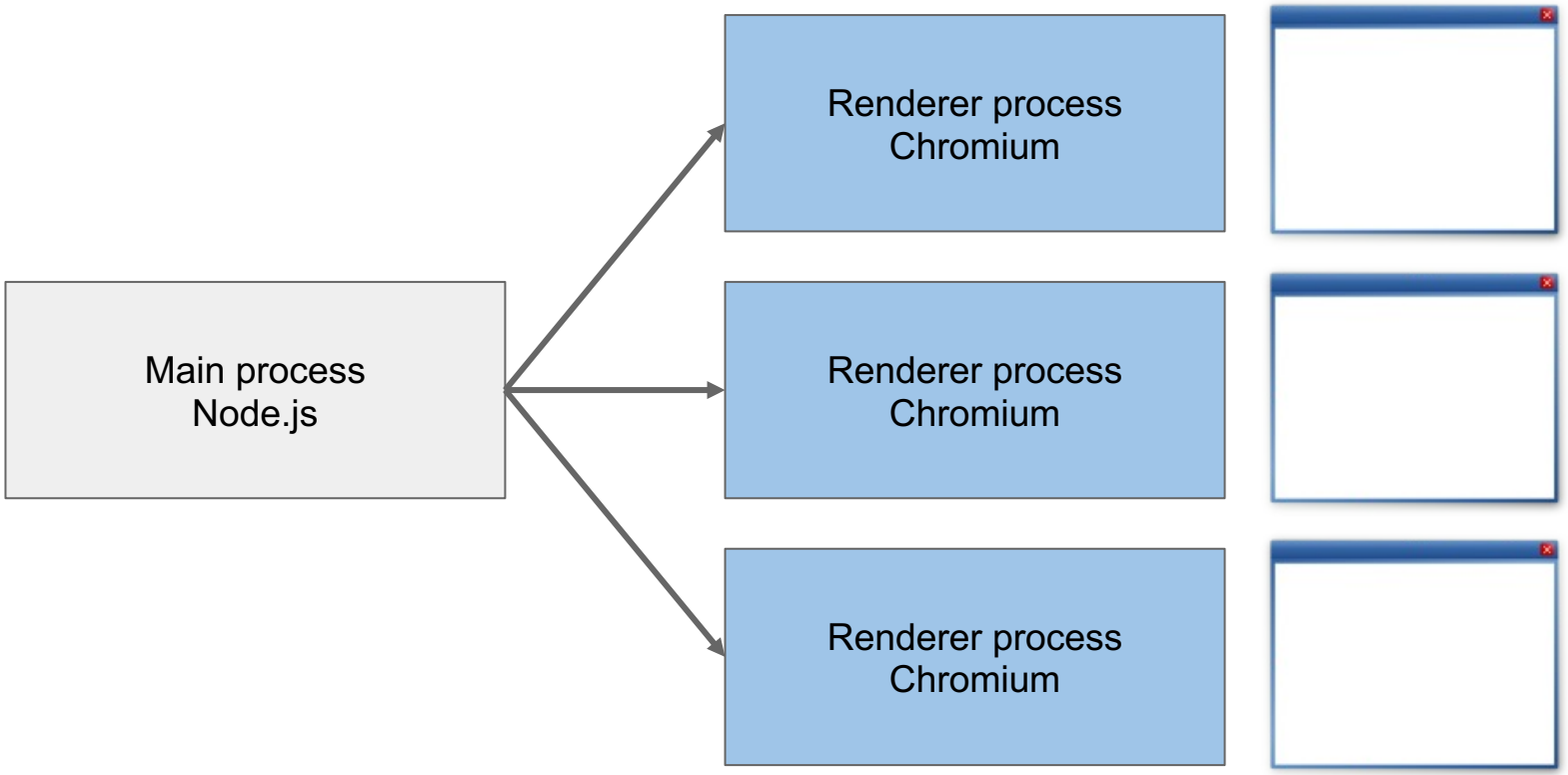
Main & renderer process

When the application is executed, the application entry point written in JavaScript runs in the so called **main process**.

The man process can instantiate one or more **`BrowserWindow`**, to create GUI windows on which HTML based content can be visualized.

Each JavaScript code defined in a BrowserWindow runs on an independent **renderer process.**

Main & renderer process



Main & renderer process

Each render process is isolated to the other render processes and the main process

Each process (renderer or main) executes JavaScript code in parallel with other processes (non blocking).

To avoid concurrency issues, it is not possible to share objects among different processes.

Electron provides an **IPC mechanism** for this kind of communication.

Main & renderer process

The main process can create renderer processes by instantiating a BrowserWindow class:

```
app.whenReady().then( () => {
let mainWindow = new BrowserWindow( {
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, './preload.js')
    }
  });
  mainWindow.loadFile("./index.html");
}).catch( (err) => {
  console.log(err);
})
```

Preload script

JavaScript code running in the render process cannot access Node.js APIs.

Preload scripts contain code that executes in a renderer process before its web content begins loading. It has access to Node.js APIs and can inject objects in the browser **window** through the **contextBridge** module.

Preload script

```
const { contextBridge } =
require('electron');


contextBridge.exposeInMainWorld('myAPI', {
    node_version:
process.versions["node"],
    chrome_version:
process.versions["chrome"],
    electron_version:
process.versions["electron"]
});
```

preload.js

```
…

<script>
Console.log(
window.myAPI.node_version )
</script>

…
```

index.html

Events

Similar to mobile apps, the app object an emit several events corresponding to the different phases of the application lifecycle

ready: emitted when the framework is initialized and ready to create new windows

window-all-closed: emitted when all the windows have been closed

Quit: emitted just before the application is terminated

Events

Some events refers to the underlying Chromium browser running in background

login: emitted when the web content requires a username/password pair for basic authentication

select-client-certificate: emitted when the HTTPS connection ask the user to select a valid certificate

Example application

IPC

Main and renderer process can communicate through two modules:

- **ipcMain**: used in the main process to receive asynchronous messages or register function invocation handles from the render process
- **ipcRenderer**: used in a render process to send asynchronous messages or invoke functions to the main process

By combining the two it is possible to let multiple render process communicate

System dialogs

Electron provides a “dialog” object to open standard dialog box provided by the OS. For instance:

- Open / save dialogs
- Message boxes

Dialogs can only be opened in the main process. Therefore, IPC must be used to create a new dialog box and to communicate the result back to the rendering process.

Angular & Electron

It is possible to transform an Angular web application to a standalone desktop one thanks to the Electron framework.

Similarly to a Cordova-based mobile application, it is sufficient to produce the html+js files with the command `$ ng build` and then open the `index.html` in a BrowserWindow

Managing additional modules

The only necessary modification it to change the tag base in the index.html file:

```
<base href="./">
```

Packaging

Electron provides a tool named **electron-packager** to create a self-contained executable for any supported platform.

Usage:

```
electron-packager . <appName>
--platform=<platform>
--arch=<architecture>
--out=<outputDirectory>
--overwrite
--icon=path/to/custom/icon
```

More info...

https://www.electronjs.org/docs/latest/tutorial/quick-start