Electron First Step

2025 年 10 月 16 日

概要

Electron を利用して、デスクトップアプリケーションを作ろう

目次

1　はじめに

1.1　読み間違えないでね

ソースコード 1　読み間違えないでね

```
1  数字: 0123456789
2  小文字:abcdefghijklmnopqrstuvwxyz
3  大文字:ABCDEFGHIJKLMNOPQRSTUVWXYZ
4
5  1:イチ
6  l：小文字のエル
7  i：小文字のアイ
8  !:ビックリマーク
9  |:バーティカルバー。Shift と￥を押したもの。
10
11 0:ゼロ
12 o：小文字のオー
13 O：大文字のオー
14
15 .:ピリオド
16 ,:コンマ
```

1.2　注意

- これから出てくるソースコードには、左に「行番号」と呼ばれる番号が出てくるけど、入力する必要ないからね。
- script タグの中で「//」で始まる文は、コメントで、プログラムは読み飛ばすよ。
- コピーできるところはコピーして効率よく入力して行こう
- 徐々に追加されていくから、量が多く見えるけど、平気だよ！
- 改行されていても、行番号が書かれていないところは、1 行だからね。表示上改行されて見えてるだけ

2　Electron とバージョンアップ

どんどんバージョンアップされています。昨年バージョンがうまく動かなくなったり等のことがよくアプリ開発では起きますので、そういうものだと思っておきましょう。

主な理由は、セキュリティ周りの仕様変更となります。2 年連続で動かなくなりました。

3　Electron のインストール

ちょっとややこしいのですが、パッケージ管理システムを複数利用してインストールします。

Homebrew　mac 用のパッケージ管理システム
npm　node.js 用のパッケージ管理システム

1. Homebrew のインストール
2. brew を使って Node.js のインストール
3. npm を使って Electron のインストール

3.1　homebrew のインストール

https://brew.sh/

にアクセスして、インストールと書いてある行をコピーしましょう。ターミナルを開いてペーストしてリターンしましょう。

XCode CommandLine Tool のインストールを聞かれた場合、メッセージに応じてタイピングしてください (多分リターンキーで OK)

3.2　Node.js のインストール

引き続いてターミナルで

```
$ brew install node
```

としましょう。

エラーっぽいのが出てきた場合には、そのメッセージに応じて対応しましょう。

- npm audit fix
- npm audit fix –force

等を求められたりします。

3.3　Electron のインストール

引き続いてターミナルで

```
$ npm install electron -g
```

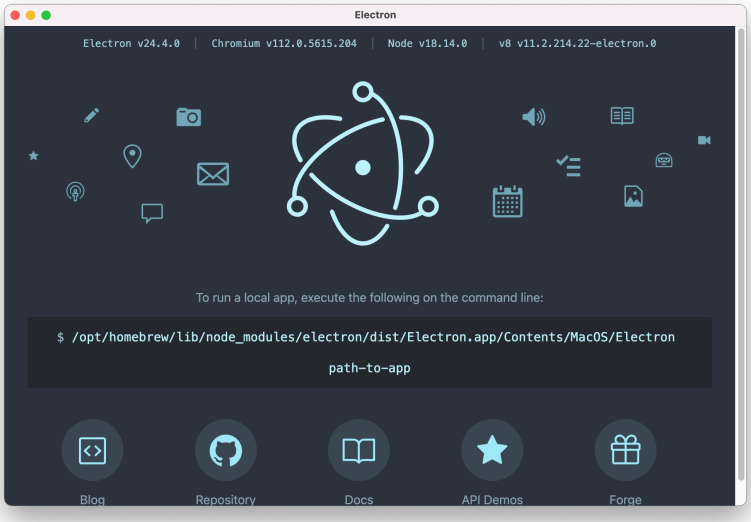
としましょう。

4　とりあえず起動してみる。

ターミナルで

$ electron

としてみましょう。
次のようなウィンドウが立ち上がれば正解です。
終了するには、ターミナル上で Ctrl-C とします。



5　やっぱり　"Hello World"

Hello World と表示するアプリケーションを作ってみましょう。

5.1　作業用フォルダとパッケージファイルの作成

今日の作業用のフォルダを「AID06」として作成しましょう。その中に「HelloWorld」というフォルダを作成しましょう。
ターミナルで$の後に"cd "と入力した後に、作成した HelloWorld フォルダをドラッグアンドドロップしましょう。すると、

cd /Users/sammy/Desktop/AID/AID06/HelloWorld

のようになるので、リターンを押しましょう。

npm init -y

とすることで package.json というファイルが作成されるはずです。VSCode で「HelloWorld」フォルダを開きましょう。

5.2 package.json

アプリ作成に必要な設定ファイルです。

”main”: ”index.js”,

と書いてある行を

”main”: ”main.js”,

と変更しましょう。一番最初に起動するファイルが書かれています。
Electron としては、文化的に「main.js」とするようです。次に、

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

と書いてあるところを

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "electron ."
},
```

としましょう。
保存を忘れずに！

5.3 main.js

それでは新規ファイルで main.js を作成してみましょう。

ソースコード 2　Hello World：main.js

```
1  const { app, BrowserWindow } = require('electron/main')
2
3  //ウインドウの作成
4  const createWindow = () => {
5    const win = new BrowserWindow({
6      width: 400,
7      height: 400,
8      webPreferences: {
9          nodeIntegration: true,
10         contextIsolation: false,
11     },
12   })
13   // webPreferences でメインプロセスとレンダラープロセスの通信をできるようにセキュリティ下げてる
14
15   // はじめにロードするファイルをindex.html に設定
16   win.loadFile('index.html')
```

```
17
18  // ウィンドウ閉じられた時の挙動
19   win.on('closed', ()=>{
20     app.quit();
21   })
22 }
23
24 // electron の準備ができた時の挙動。基本、createWindow()を呼び出す
25 app.whenReady().then(() => {
26   createWindow()
27
28   // electron がアクティブになった時
29   app.on('activate', () => {
30     if (BrowserWindow.getAllWindows().length === 0) {
31       createWindow()
32     }
33   })
34 })
35
36 // ウィンドウが全部閉じられた時の挙動
37 app.on('window-all-closed', () => {
38   if (process.platform !== 'darwin') {
39     app.quit()
40   }
41 })
```

保存を忘れないくださいね。

5.4 index.html

17 行目でウィンドウに表示する内容として「index.html」とされています。これを作成してみましょう。

ソースコード 3　Hello World：index.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8" />
5      <!-- セキュリティ回避 -->
6      <meta http-equiv="Content-Security-Policy"
7          content="script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline
               ';">
8      <title>Hello from Electron renderer!</title>
9      <link rel="stylesheet" href="css/style.css">
10   </head>
11   <body>
12     <h1>Hello from Electron renderer!</h1>
13         <p>
14             We are using node
15             <script>document.write(process.versions.node)</script>,
```

```
16          </p>
17          <p>
18              Chrome
19              <script>document.write(process.versions.chrome)</script>,
20          </p>
21          <p>
22              and Electron
23              <script>document.write(process.versions.electron)</script>
24          </p>
25
26    </body>
27  </html>
```

- procdess.versions.node

は Node.js で定義されていて

- procdess.versions.chrome
- procdess.versions.electron

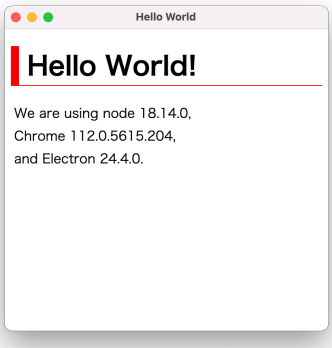
は、Electron で拡張された機能となります。
　meta タグにてセキュリティの程度をゆるめています。css ファイルはまだないですが、link 先に指定しています。

5.5　実行してみよう

ターミナルで

npm start

とすると、ウィンドウが次のよう表示されるはずです。(あ、この画像すでに css ついてる... 気にしない...)



また、Dock で Electron のアプリ、として起動していることも確認しましょう。
ちょっと説明します。package.json で

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "electron ."
```

```
  },
```

としたため、”npm start”と入力すると、この start 部分が呼ばれて、”electron .”が実行されるわけです。
”electron .”は、このフォルダの中で electron を実行せよ、という意味になります。
メニューから終了しましょう。

5.6 css を追加してみよう

「HelloWorld」フォルダの中に「css」フォルダを作り、その中に「style.css」を作成してみましょう。
style.css を以下のようにしましょう。

ソースコード 4　Hello World：css/style.css

```
1 h1 {
2     border-left: 10px solid red;
3     border-bottom: 1px solid red;
4     padding-left: 10px;
5 }
6 p {
7     margin: 4px;
8 }
```

もう一度ターミナルで「npm start」で実行してみましょう。反映されてますね。Web と変わらず CSS ファイルも利用することができることがわかりました。

ついでに Option+Command+I をすると、デベロッパーツールが使えることがわかります。Webkit(chrome で利用している blink も webkit の派生) の技術を使っているため、ほとんどブラウザと同じ、ということになります。

Command-Q で終了できますが、ターミナルに$マークが見えてない場合には、きちんと終了できていないため、Ctrl-C で終了することを覚えましょう。

6 時計作るぞ！

6.1 作業用フォルダとパッケージファイルの作成

「AID06」の中に「DigitalClock」というフォルダを作成しましょう。

ターミナルで$の後に”cd ”と入力した後に、作成した DigitalClock フォルダをドラッグアンドドロップしましょう。すると、

cd /Users/sammy/Desktop/AID/AID06/DigitalClock

のようになるので、リターンを押しましょう。

npm init -y

とすることで package.json というファイルが作成されるはずです。VSCode で「DigitalClock」フォルダを開きましょう。

6.2 package.json

アプリ作成に必要な設定ファイルです。

”main”: ”index.js”,

と書いてある行を

”main”: ”main.js”,

と変更しましょう。一番最初に起動するファイルが書かれています。
繰り返しになりますが、Electron としては、文化的に「main.js」とするようです。次に、

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

と書いてあるところを

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "electron ."
},
```

としましょう。
保存を忘れずに！

6.3 main.js

HelloWorld のものをそのまま複製しましょう。

```
width: 400,
height: 400,
```

を

```
width: 200,
height: 100,
```

に変更しておきましょう。

6.4 index.html

新規ファイルとして作成しましょう。

ソースコード 5　DigitalClock：index.html

```
1 <!DOCTYPE html>
2 <html>
3   <head>
```

```
4     <meta charset="UTF-8" />
5
6     <!-- セキュリティ回避 google font 利用できるように追加-->
7     <meta http-equiv="Content-Security-Policy"
8       content="script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'
            https://fonts.googleapis.com; font-src 'self' https://fonts.gstatic.com;">
9
10    <title>Hello from Electron renderer!</title>
11    <link rel="stylesheet" href="css/style.css">
12  </head>
13  <body>
14    <div id="digital_clock"></div>
15    <script src="clock.js"></script>
16  </body>
17 </html>
```

6.5 clock.js

前の 9 行目で clock.js が呼び出されているため、それを作成しましょう。

ソースコード 6　DigitalClock：clock.js

```
1  // 時計の描画処理をスタート
2  clock();
3
4  function clock () {
5      // 現在日時を取得
6      let d = new Date();
7
8      // デジタル時計を更新
9      updateDigitalClock(d);
10
11     // 次の「0ミリ秒」に実行されるよう、次の描画処理を予約
12     let delay = 1000 - new Date().getMilliseconds();
13     setTimeout(clock, delay);
14 }
15
16 function updateDigitalClock (d) {
17     const AA_str = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"];
18     let YY = d.getFullYear().toString().slice(-2);
19     let MM = d.getMonth() + 1;
20     let DD = d.getDate();
21     let AA = d.getDay();
22     let hh = d.getHours();
23     let mm = d.getMinutes();
24     let ss = d.getSeconds();
25
26     // 桁あわせ
27     if(MM < 10) { MM = "0" + MM; }
```

```
28      if(DD < 10) { DD = "0" + DD; }
29      if(hh < 10) { hh = "0" + hh; }
30      if(mm < 10) { mm = "0" + mm; }
31      if(ss < 10) { ss = "0" + ss; }
32
33      let text = YY + '/' + MM + '/' + DD + ' (' + AA_str[AA] + ')<br>' + hh + ':'
            + mm + ':' + ss;
34      document.getElementById("digital_clock").innerHTML = text;
35  }
```

「npm start」でとりあえず、時計が動いていることがわかると思います。

6.6 透過ウィンドウへの変更

main.js の win=new BrowserWindow 部分を以下のように 3 行追記しましょう。

ソースコード 7　DigitalClock：main.js

```
1       win = new BrowserWindow({
2           width: 200,
3           height: 100,
4           transparent: true, // ウィンドウの背景を透過
5           frame: false, // 枠の無いウィンドウ
6           resizable: false, // ウィンドウのリサイズを禁止
7           webPreferences: {
8               nodeIntegration: true, //Electron6 から必要らしい
9               contextIsolation: false, //Security 的には良くないらしいが...
10          }
11      })
```

6.7 css/clock.css の追加

このままだと、ドラッグも何もできないため、背景を CSS でいじりましょう。css フォルダを作成してその中に clock.css を追加します。

index.html で css/clock.css のリンク追加を忘れないように!

ソースコード 8　DigitalClock：css/clock.css

```
1   body {
2       background-color: rgba(24, 24, 24, .7);
3       color: #fff;
4       -webkit-app-region: drag;
5       -webkit-user-select: none;
6       user-select: none;
7   }
```

ちょっと説明します。

背景色/背景画像　透過ウィンドウでは「何もない部分」はクリック出来ないため、背景を指定します rgba(r, g, b, a) による色指定を行うことで、透明度を持った背景色が指定できます

-webkit-app-region: drag;　要素をウィンドウのタイトルバーのように扱う指定ですいずれかの要素にこの指定を行わないと、ドラッグによるウィンドウ移動が出来ません

-webkit-app-region: no-drag;　drag を指定した要素内にボタンなど操作可能な要素を配置する場合、no-drag を指定して上書きします

-webkit-user-select: none;　テキストの選択を無効化します主にインタフェース部分に指定します

-webkit-となっているのは「ベンダープレフィックス」と言って、Webkit(派生の blink も) のみで利用できる CSS となっています。

index.html から css ファイルへのリンクを忘れないようにしましょう。

6.8 css の調整

見た目をさらに調整するために、以下のようにしましょう。

ソースコード 9　DigitalClock：css/clock.css

```
1  @import url('https://fonts.googleapis.com/css2?family=Iceland&display=swap');
2
3  body {
4      overflow: hidden;
5      margin: 0;
6      padding: 0;
7      border: 5px solid rgb(42,42,42);
8      background-color: rgba(24, 24, 24, .7);
9      box-shadow: 0 0 8px 3px #000 inset;
10     color: #fff;
11     -webkit-app-region: drag;
12     -webkit-user-select: none;
13     user-select: none;
14 }
15
16 #digital_clock {
17     font-family: "Iceland", sans-serif;
18     font-size: 25px;
19     line-height: 40px;
20     margin-top: 9px;
21     text-align: center;
22     color: #fff;
23     text-shadow: 1px 1px 3px #000;
24 }
```

@import 文は、Google Fonts というサービスを利用して、フォントを利用できるようにしています。

詳しくは

https://saruwakakun.com/html-css/basic/google-fonts https://peraichi.com/univ/20220502

をみましょう。

7　アプリケーションとして配布できるようにしよう

7.1　これまで

ターミナルで「 npm start」で起動してるので、アプリケーションっぽいんだけど、アプリケーションっぽくないですね。配布できるようにしましょう。

7.2　electron-builder

ターミナルで

npm install -D electron-builder

としてください。1 分くらいかかりますが、「node_modules」というフォルダが増えましたね。
後、アプリ作るのに、node_modules の中に必要っぽいので

npm install -D electron

もしましょう。

7.3　package.json

```
"devDependencies": {
  "electron": "^38.2.2",
  "electron-builder": "^26.0.12"
}
```

の後に (バージョン情報は変わる可能性あり) 以下のように 6 行追記してください。「,」を忘れずに

```
"devDependencies": {
  "electron": "^38.2.2",
  "electron-builder": "^26.0.12"
},
"build": {
  "appId": "jp.test.app1",
  "mac": {
    "target": "dmg"
  }
}
```

7.4　build

実際にファイルを構築することをビルド、と言います。
ターミナルで

```
./node_modules/.bin/electron-builder --mac --universal
```

としましょう。うまくいけば、dist フォルダができて、その中に配布用の dmg ファイルができるはずです。

mac には Intel, AppleSilicon の二つの種類の CPU がありますが、そのどちらでも動くアプリケーションを作成するために「–universal」としています。

windows の場合には次を参考にしてください。(うまく行くかな...)

https://maku.blog/p/2tcs8n2/

7.5 セキュリティについて

アプリ開発時にはセキュリティについて考慮する必要があります。今回のサンプルでは煩雑になるため手抜きしています。詳しくは次を参考にしてください。

https://gist.github.com/umamichi/5d52367235c98425e9d3fa4439d35046
https://zenn.dev/sprout2000/books/6f6a0bf2fd301c/viewer/13340

8 まとめ

これまで学んできた、Canvas,WebGL も JavaScript なわけですから、Electron を利用することでアプリケーションを開発することができたのが理解できたでしょうか？

アイコンを変更したり、メニューバーを改造したり、普通のアプリ開発で必要なことは相当できるみたいです。

興味を持った人は、以下をみてみましょう。

https://qiita.com/nyanchu/items/9a1c910bbca55e9d2f3c

また、アナログ時計を作ってみたい人は、以下のページを見てみてください。

https://qiita.com/Yuta_spade/items/2493c05cd868ea5f2677

P.S.

バージョンアップすると、動かなくなるから、それの対処方法に今年も少しハマったー

以上