Electron を用いてアプリケーションを開発

Node.js のインストール

まず，Node.js をインストールする．
その際，次のサイトを参考に行った．

- Node.js をインストールする

Electron のインストール

1. package.json の作成

作業ディレクトリを作成し，そこで `npm init -y` を実行する．
すると，ディレクトリ内に以下のような `package.json` ファイルが作成される．

```
{
  "name": "analysis",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "dependencies": {},
  "devDependencies": {},
  "author": "",
  "license": "ISC",
  "description": ""
}
```

`name`, `version`, `description`, `license` などのデータは単なるパッケージのメタデータであり，
パッケージを公開するつもりがないなら特に気にする必要はない．
機能的に重要なのは `bin`, `main`, `dependencies`, `devDependencies`, `scripts` である．

dependencies & devDependencies

`dependency` とはそのパッケージが依存する別のパッケージであり，パッケージ名とその version で構成される．
さらに，`dependencies` には実行に必要なパッケージ，`devDependencies` には開発やテストにのみ必要なパッケージが記録される．この 2 つの機能的な違いは，あるパッケージ A を `dependency` としてインストールするときにデフォルトでは A の `dependencies` はインストールされるが A の `devDependencies` はインストールされない．

次に，コマンドプロンプト上で `npm install --save electron` を実行する．
ここで，`npm` とは**Node.js のパッケージ管理ツール**を使用する際のコマンドである．
詳細は、『NPM と package.json を概念的に理解する』
また，`--save` オプションを付与すると `package.json` の `dependencies` に依存関係が追記される．`dependencies` には公開する際に必要なパッケージを記録する．一方，`-dev` オプションを付与すると `package.json` の `devDependencies` に依存関係が追記される．この `devDependencies` には開発に必要なパッケージを記録する．

詳細は、『npm 入門』を参照．

install した package はカレントディレクトリ上の保存される．そのため，カレントディレクトリ を git で 管理している場合には，package を git で同期しないように `gitignore` に追記する必要がある．

```
$ カレントディレクトリ/node_modules/
```

Electron サンプル作成

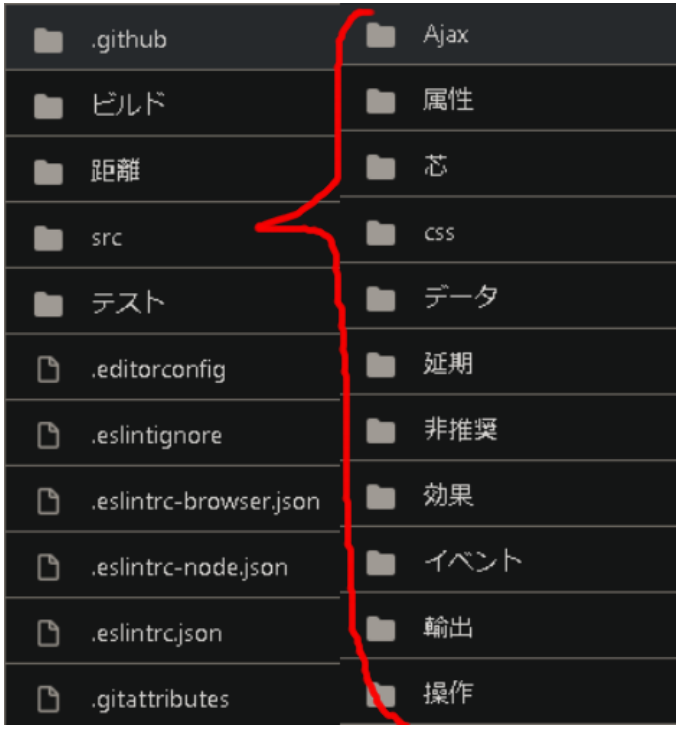
では，実際に `Electron` でアプリを開発していく．

現在，ディレクトリは以下のような構成となっている．

```
---
./
 |
 |-package.json // 環境構築用
```

そこで，新しくカレントディレクトリに `src` ディレクトリを追加する．この `src` とは `source` の略であり，jQuery の github などを見てみると、 `src` ディレクトリの内部にプログラムで使用するパッケージが全て入っていることがわかる．

詳細は，『GitHub の Web 系プロジェクトのフォルダ構成を調べてみる』，『ディレクトリ名 src とかの謎と作法』を参照．



メインプログラム用の Package.json を作成

次に，作成した `src` ディレクトリ内部に `package.json` ファイルを作成する．こちらには，**エントリーポイント** (プログラムの実行段階において、プログラムやルーチンの実行する開始位置のこと) **となる JavaScript ファイルを指定する**．また，アプリケーション名と version を指定しておくと，後ほど管理が簡単になる．

```
{
  "main": "main.js"
}
```

メインプログラムを作る

では，エントリーポイントとなる JavaScript ファイルを作成していく．先ほどと同じ `src` ディレクトリ内に `main.js` ファイルを作成し．以下のプログラムを記述する．

```
// アプリケーション作成用のモジュールを読み込み
const {app, BrowserWindow} = require('electron');

// メインウィンドウ
let mainWindow;

function createWindow() {
  // メインウィンドウを作成します
  mainWindow = new BrowserWindow({
    webPreferences: {
      nodeIntegration: true,
    },
    width: 800, height: 600,
  });

  // メインウィンドウに表示するURLを指定します
  //  （今回はmain.jsと同じディレクトリのindex.html）
  mainWindow.loadFile('index.html');

  // デベロッパーツールの起動
  mainWindow.webContents.openDevTools();

  // メインウィンドウが閉じられたときの処理
  mainWindow.on('closed', () => {
    mainWindow = null;
  });
}

//  初期化が完了した時の処理
app.on('ready', createWindow);

// 全てのウィンドウが閉じたときの処理
app.on('window-all-closed', () => {
  // macOSのとき以外はアプリケーションを終了させます
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
// アプリケーションがアクティブになった時の処理(Macだと、Dockがクリックされた時)
app.on('activate', () => {
  // メインウィンドウが消えている場合は再度メインウィンドウを作成する
  if (mainWindow === null) {
    createWindow();
  }
});
```

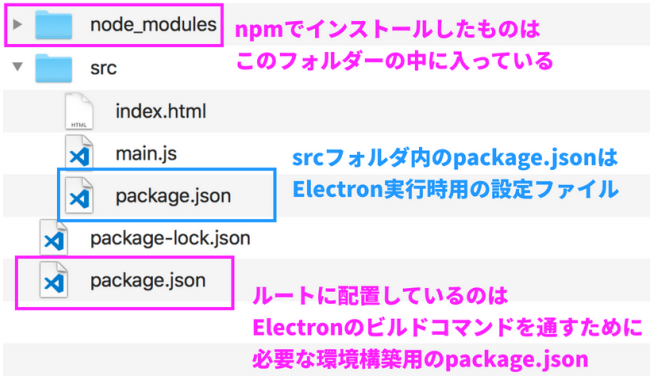
メインウインドウで文字などを表示するための HTML ファイル

`JavaScript` ファイルによってウインドウの制御部分は設計できた．次に，ウインドウ上に文字などを表示するための `HTML` ファイルを作成する．

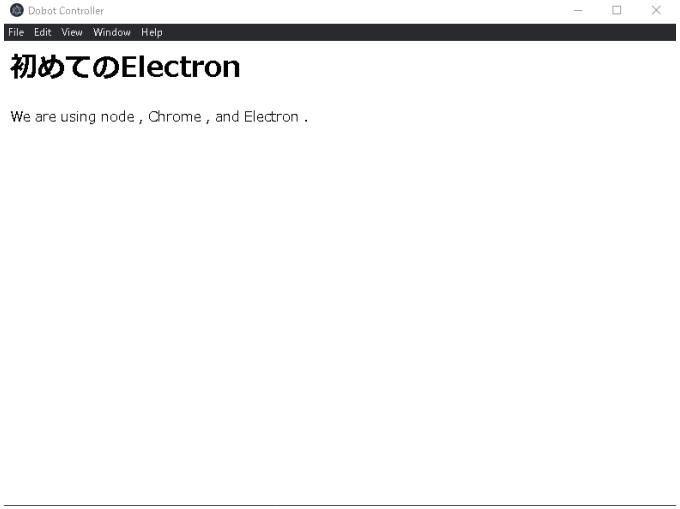
```
<html>
<head>
  <meta charset="UTF-8">
  <title>Hello World!</title>
</head>

<body>
  <h1>初めてのElectron</h1>
  We are using node <script>document.write(process.versions.node)</script>,
  Chrome <script>document.write(process.versions.chrome)</script>,
  and Electron <script>document.write(process.versions.electron)</script>.
</body>
</html>
```

アプリケーションの実行

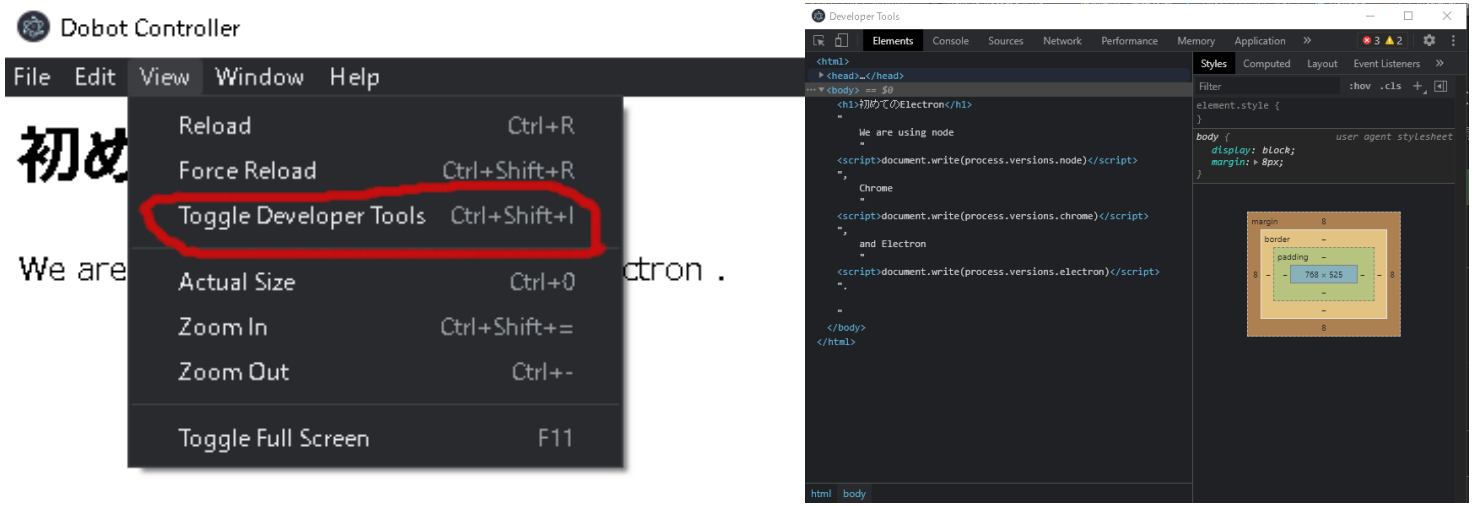
まず，今まで行った作業によって，ディレクトリとファイルは以下のような配置になっている．



実際に起動したアプリケーションは右図のようになる．



さらに，表示された画面の `View` から `Toggle Developer Tools` を押すことで，デベロッパーツール画面に移ることができる．



しかし，これでは画面上に DevTool しか表示されないため，実際のウインドウを見ながらデバッグすることができない．そこで，以下の画像の通りにボタンを押す．これによって，DevTool と ウインドウを別々のウインドウとして表示することができる．