第３章

タスクランナー

HTML/CSS/Jsなど各言語で構築する際に、作業の効率化
品質の一定化など意識した環境づくりの為のセッティングを行います。

node.js

ネットワークアプリケーションを利用した開発環境のための環境作り

古い環境の削除

すでにインストールしている人の中で、古い状態のデータが入っている場合、一度アンイストールが必要です。
過去今まで一度も環境構築していない人は読み飛ばして構いません。

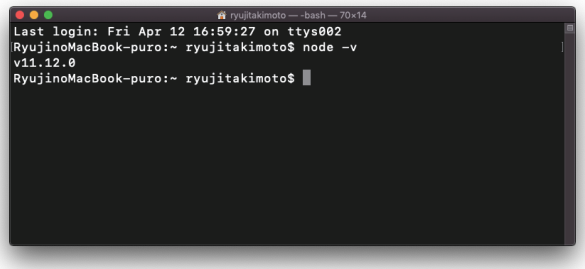
node環境の確認・アンイストール

プレーンな状態でnodeをインストールしている場合、各プロジェクト環境でnode環境のバージョンが違うといったケースでは、nodeを各バージョンに合わせて入れ直す必要があり、管理が大変です。

node のバージョン確認（ターミナルから実行）

```
node -v
```

個人で作業する分には問題ありませんが、グループでの作業や、途中でプロジェクトにアサインする場合など環境の統一が必要になります。



v11.12.0 これが現在インストールされているnodeのバージョンです。

既存のpkgで入れたnode.jsの削除

nodeのバージョンを管理するnodebrewをインストールし、その上でnode環境を構築します。すでにプレーンな状態でnodeをインストールしている場合、一度アンイストールしなければいけません。

/var/db/receipts/ 内のnode.js 類の削除

```
#nodeファイルの検索
which node

#nodeファイルの削除
sudo rm -rf /usr/local/bin/node    <    ファイルパスは各自確認
```

削除を行う際は、過去のプロジェクトに影響が出る為、各自責任の元行ってください。

npmコマンドの削除

```
sudo rm -rf ~/.npm
```

sudoコマンドはスーパーユーザー権限で実行する関係上、PCのユーザーパスワードが必要です。通常のパスワードのように、入力しても●●●と表示されませんが、入力後エンターキーで実行が可能です。

通常のインストールであれば、これでアンイストールが完了です。

colum　nodeをディレクトリ管理する場合はnodenvで管理

資料では、nodebrewを使ってnodeを管理していますが、ディレクトリでnodeを管理出来ません。ディレクトリ単位でnodeを管理したい場合は別にnodenvという物が存在します。

nodenv
https://github.com/nodenv/nodenv

（新規はここから）ターミナルでbash設定ファイルの作成

インストール環境作成のために、ターミナルからbash設定ファイルを作成します。bashとは、LinuxやMacでのターミナルコマンドライン（シェルスクリプト）の方法です。※Macの標準コマンド

bash設定ファイル作成場所

```
~/.bash_profile
```

zshでの設定ファイル作成場所

```
~/.zshrc
```

自分のmac環境でどちらか一方の設定のみ行う。

nodebrewのインストール

設定ファイルを作成することができたら、nodebrewのインストールです。nodebrew自体はnodeのバージョンを管理するためのサービスです。授業ではバージョンは最新版を固定で使用しますが、過去のプロジェクトや、バージョンの違うnodeを触りたい場合は、別のバージョンをインストールし使用する事が可能です。

nodebrew のインストール

```
curl -L git.io/nodebrew | perl - setup
```

nodebrew公式GitHub
https://github.com/hokaccha/nodebrew

nodebrewのパス設定

インストールが環境したら、nodebrewを使用するためのパス設定を行います。

nodebrew のパス設定 ~/.bash_profile or ~/.zshrc

```
export PATH=$HOME/.nodebrew/current/bin:$PATH
```

nodebrew のパス設定 ~/.zshrc

```
source ~/.bash_profile
```

パスの設定が完了したら、nodebrewが利用出来るかチェック

nodebrew のパス設定

```
#パスが通ったか確認
which nodebrew
#nodebrewの使用確認
nodebrew help
```

```
RyujinoMacBook-puro:~ ryujitakimoto$ nodebrew help
nodebrew 1.0.1

Usage:
    nodebrew help                        Show this message
    nodebrew install <version>           Download and install <versio
n> (from binary)
```

nodebrewのバージョンが表示されるので、無事インストールが出来たことが分かります。

node.jsのインストール

nodebrewがインストール出来たら、nodebrew上にnode.jsのインストールをします。各バージョン毎にインストールが可能なので、必要に応じてバージョンをインストールしましょう。

node.js （最新版）のインストール

```
nodebrew install-binary latest
```

安定版の場合： stable
バージョン指定をする場合：v10.0.0 の様にバージョンを記入する。

nodebrew 上でnode.jsの切り替え

```
#最新版を使用する場合
nodebrew use lateest

#安定版を使用する場合
nodebrew use stable

#バージョン(v10.0.0を使う場合)を指定して使用する場合
nodebrew use v10.0.0
```

node.jsのインストール・使用については以上です。以降は、node.js上で動くタスクランナーに移ります。

タスクランナー

作業環境効率化のための支援ツール

Webサイト構築における自動処理(タスクランナー)

Web制作の開発の中（フロントエンドエンジニア）で、各ブラウザ（種類・バージョン）への対応や、複数人での開発の中での品質の均一など、本来時間をかける必要が無いまたは時間をかけていられないもの。長い時間開発を行う中で、小さな時間が積み重なるような煩雑な事が多々あります。
時代の流れと共に不要になる内容もある中で、メインのコーディングに集中するために考えられたものがタスクランナーといいます。

gulp

webpack

Grunt

代表的なタスクランナー

主なタスクランナーは左の３つがよく使われているものになります。
nodeの環境に自動処理機能（タスクランナー）を入れて、面倒なタスク（圧縮・結合・prefixなどなど）を実行させます。
授業では、gulpベースで環境を整えますが、webpackやgruntの環境も自分で作成して構いません。
※ どれか１つでOKです

gulpのインストール（グローバル）

まずはgulpをインストールします。グローバル環境（PC全体）にインストールします。

gulpのグローバルインストール

```
sudo npm install gulp -g
```

```
gulp -v
```

グローバルインストール（-g）で全体に対してインストールを行う。インストールが完了したらバージョン確認（-v）をして、正しくインストールが出来たか確認をしておく。

gulpの環境構築

gulpのインストールが終了したら、gulpを利用した環境構築を行います。ターミナルから制作を行うディレクトリを移動します。

ディレクトリの移動

```
# ディレクトリの移動方法
cd ディレクトリ名

# 現在位置の確認
ls
```

HTML3の制作環境まで（cd）を使いながら移動します。

package.jsonファイルの作成

```
npm init
```

環境を明記するためのpackage.jsonファイルを作成します。

gulpの環境構築

ローカル環境に必要なデータをインストールします。

gulpのインストール（ローカル）

```
sudo npm install gulp -d
```

es6-promiseのインストール

gulpの拡張機能を利用する為の互換性を保つためのpromiseファイルを読み込み

es6-promise

```
sudo npm install es6-promise -d
```

gulp-sassのインストール

gulpを利用したsassファイルを作成する為の拡張機能のインストール

gulp-sass

```
sudo npm install gulp-sass -d
```

gulp-autoprefixerのインストール

ベンダープレフィックスを設定するgulp-autoprefixerをインストール

gulp-pleeease

```
sudo npm install gulp-autoprefixer -d
```

gulp-sourcemapsのインストール

コンパイルする前のファイル場所を確認するためのソースマップファイルのインストール

gulp-sourcemaps

```
sudo npm install gulp-sourcemaps -d
```

gulp-plumberのインストール

コンパイルエラー時に動作を停止するためのgulp-plumberをインストール

gulp-plumber

```
sudo npm install gulp-plumber -d
```

gulp-notifyのインストール

デスクトップへのポップアップをするためのgulp-notifyをインストール

gulp-notify

```
sudo npm install gulp-notify -d
```

gulp-pugのインストール

HTML拡張機能のpugファイルを作成するためのgulp-pugをインストール

gulp-pug

```
sudo npm install gulp-pug -d
```

授業では上記の拡張機能を必要最低限とし、インストールを行います。個人で制作環境を作成する場合は、webpackなどの環境も自分で作成してみると良いです。
拡張機能をインストールすることが出来たら、次にタスクランナーの設定ファイル（gulpfile.js）の作成です。

gulpfile.jsの作成

gulpfile.jsを作成し、設定項目を記入していきます。
このファイルは、gulpのタスク定義をするのに使用し、先程インストールしたes6-promiseを定義ファイル内に記入することで、gulpにインポートすることができます。

gulpタスク環境の枠作成

プロジェクトフォルダの直下に「gulpfile.js」ファイルを作成。先程インストールした拡張機能の読み込み・タスク処理実行の枠を作成します。

gulpfile.js

```
//strictmodeでのjsファイルを実行
'use strict';

//moduleの追加
require('es6-promise').polyfill();
const gulp = require('gulp');
const sass = require('gulp-sass');
const autoprefixer = require('gulp-autoprefixer');
const sourcemaps = require('gulp-sourcemaps');
const plumber = require('gulp-plumber');
const notify = require('gulp-notify');
const pug = require('gulp-pug');

const { series,task } = gulp;

//taskの実行
task('sass',(done)=>{
  console.log("sassの実行");
  done();
})

task('default',series('sass'));
```

保存を環境したら。ターミナルで実行します。

taskの実行（ターミナル）

```
gulp
```

```
kadai — -bash — 88×8
RyujinoMacBook-puro:kadai ryujitakimoto$ gulp
[16:07:46] Using gulpfile /Volumes/vol/site/test.local/2019/html3/kadai/gulpfile.js
[16:07:46] Starting 'default'...
[16:07:46] Starting 'sass'...
taskの実行
[16:07:46] Finished 'sass' after 1.1 ms
[16:07:46] Finished 'default' after 4.17 ms
RyujinoMacBook-puro:kadai ryujitakimoto$
```

gulpfile.jsのsassタスクで入力したconsoleの文字がターミナル上で表示されました。（実行できた）

出力データの設定・タスク処理の枠組み作成

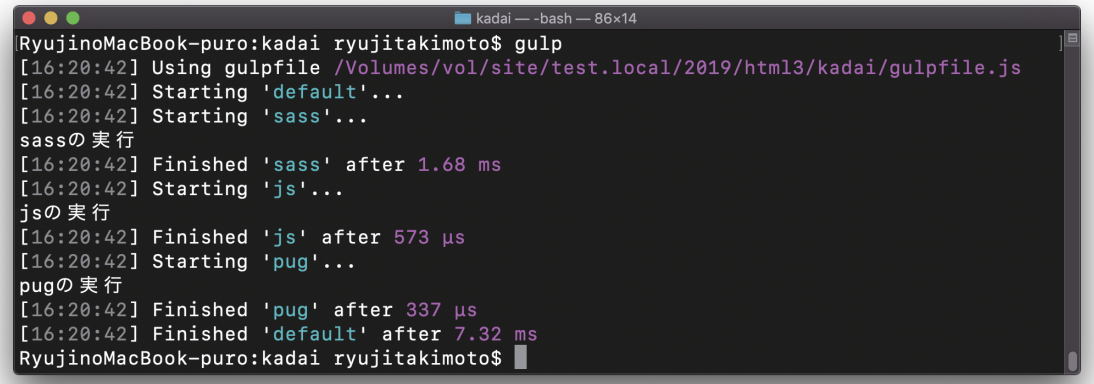
読み込みデータ・出力データの設定や各種タスク設定を追記します。

gulp-pug

```
～略
const { series,task,src,dest,watch } = gulp;

// in/output dir path
const cssSrcPath = './src/sass';
const cssDestPath = './css';
const jsSrcPath = './src/js';
const jsDestPath = './js';
const pugSrcPath = './src/pug';
const pugDestPath = './';

//taskの実行
task('sass',(done)=>{
  console.log("taskの実行");
  done();
})
//taskの実行
task('sass',(done)=>{
  console.log("sassの実行");
  done();
})
task('js',(done)=>{
  console.log("jsの実行");
  done();
})
task('pug',(done)=>{
  console.log("pugの実行");
  done();
})

task('default',series('sass','js','pug'));
```

CSS作業フォルダの場所
CSS出力フォルダの場所
Js作業フォルダの場所
Js出力フォルダの場所
HTML作業フォルダの場所
HTML出力フォルダの場所

各タスクの枠組みが作成出来たら、一度実行（gulp）を行います。



各タスクの種類毎（CSS/Js/HTML）別にタスクが実行された。
次は、各言語別にタスク内容を記入していきます。

sassタスクの設定

sass（css）のタスク内容を追記します。

task sassの指定

```
task('sass',(done)=>{
  console.log("sassの実行");
  src(cssSrcPath + '/*.scss' )
    .pipe(plumber({
        errorHandler: notify.onError('Error: <%= error.message %>')
    }))
    .pipe(sass({
        outputStyle:'expanded',
        compass : true
        }).on('error',sass.logError))
    .pipe(sourcemaps.write())
    .pipe(autoprefixer({browsers: ["last 3 versions", "ie >= 11", "Android >= 4","ios_saf >= 10"]}))
    .pipe(dest(cssDestPath));
  done();
})
```

jsタスクの設定

js（javaScript）のタスク内容を追記します。

task jsの指定

```
task('js',(done)=>{
  console.log("jsの実行");
  src(jsSrcPath + '/*.js' )
  .pipe(plumber({
      errorHandler: notify.onError('Error: <%= error.message %>')
  }))
  .pipe(dest(jsDestPath));
  done();
})
```

pugタスクの設定

pug（HTML）のタスク内容を追記します。

gulp-pug

```
task('pug',(done)=>{
  console.log("pugの実行");
  src([`${pugSrcPath}/*.pug` , `!${pugSrcPath}/_*.pug`])
  .pipe(plumber({
      errorHandler: notify.onError('Error: <%= error.message %>')
  }))
  .pipe(pug({
      pretty: true
  }))
  .pipe(dest(pugDestPath));
  done();
})
```

各タスクを実行してエラーがあるかどうか確認しましょう。

watchタスクの設定

毎回実行するのは手間がかかるので、保存と同時に実行するwatchタスクを最後に設定します。

watch task

```
task('watch',(done)=>{
  console.log("watchの実行");
  watch(`${cssSrcPath}/*.scss`,task('sass'));
  watch(`${jsSrcPath}/*.js`,task('js'));
  watch(`${pugSrcPath}/*.pug`,task('pug'));
  done();
})
```

watchタスクが記入できれば、最後にgulpをwatch状態（ファイルの監視）にします。

watch

```
gulp watch
```

ファイルを保存する度にgulpが実行できれば、環境設定の完了です。