Angular

[TypeScript][Google Cloud Platform][Bootstrap][Flask]

Angular CLI

インストール

```
>npm install -g @angular/cli
```

アプリケーションの生成

```
> ng new myapp
```

アプリケーションの実行

```
> ng serve
```

Angular CLI の主なコマンド

・https://github.com/angular/angular-cli/wiki

概要	コマンド
app アプリを生成	ng new app
ひな形の生成	ng generate ...
ビルドして起動	ng serve
ビルド	ng build
ユニットテスト	ng test
E2E テスト	mg e2e
i118n メッセージを抽出	ng xi18n
指定されたキーワードで検索	ng doc keyword
TSLint によるコードチェック	ng lint
現在の設定を取得	ng get key
指定されたキー / 値を設定	ng set key=value
Angular CLI のバージョン	ng version

ng generate サブコマンド

要素	コマンド
モジュール	ng g moduole hoge
コンポーネント	ng g component hoge
ディレクティブ	ng g directive hoge
パイプ	ng g pipe hoge

サービス	ng g service hoge
ガード	ng g guard hoge
クラス	ng g class hoge
インターフェース	ng g interface hoge
列挙	ng g enum hoge

ngx-bootstrap

・https://valor-software.com/ngx-bootstrap
・Bootstrap を Agular アプリケーションから利用

Angular-CLI から利用

・https://github.com/valor-software/ngx-bootstrap/blob/development/docs/getting-started/ng-cli.md

インストール

```
npm install ngx-bootstrap bootstrap --save
```

src/app/app.module.ts の編集

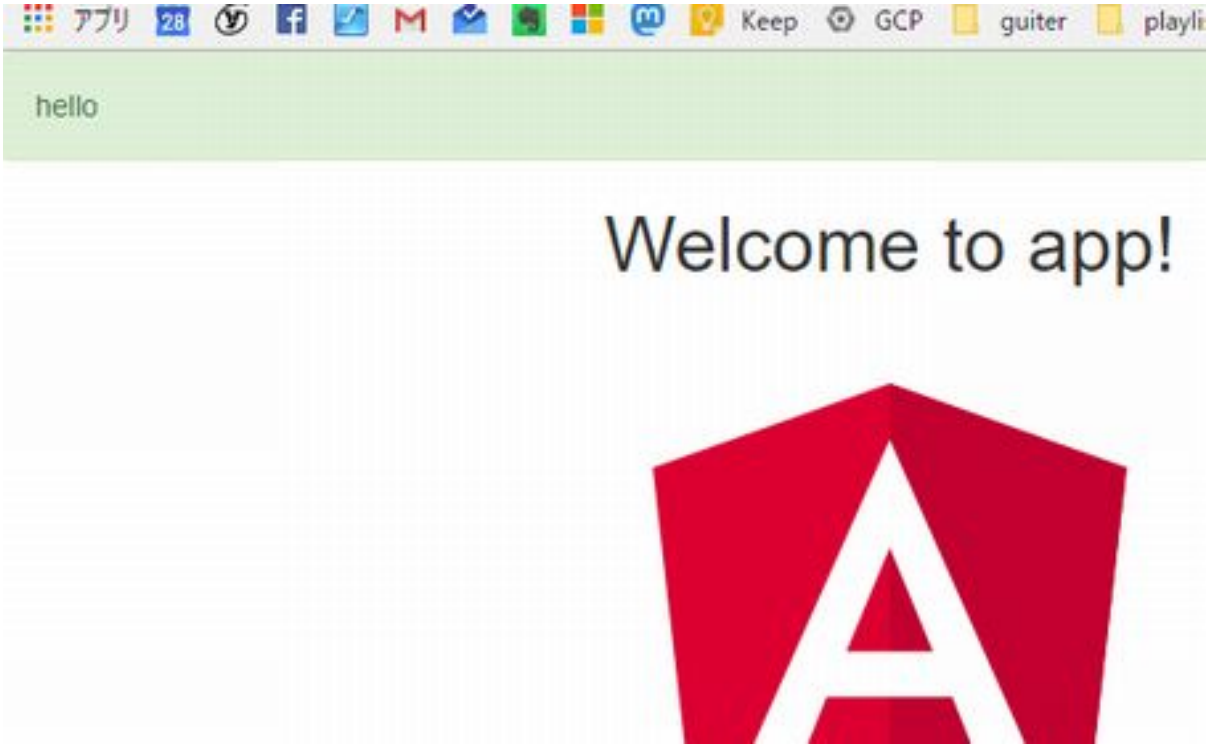
```
import { AlertModule } from 'ngx-bootstrap';
      :
@NgModule({
      :
   imports: [AlertModule.forRoot(), ... ],
      :
})
```

.angular-cli.json に以下を追加

```
"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.css"
],
```

src/app/app.component.html に以下を追加

```
<alert type="success">hello</alert>
```



Angular Material

・https://material.angular.io/

非同期通信

app.module.ts

```
import { HttpModule } from '@angular/http';
    :
@NgModule({
    :
  imports: [
    :
    HttpModule,
  ],
    :
})
```

データバインディング

構文

データ方向	種類	記法
コンポーネント -> ビュー	補間	{{...}}
コンポーネント -> ビュー	プロパティ / 属性バインディング	[property]='value'
ビュー -> コンポーネント	イベントバインディング	(event)='handler'
コンポーネント <-> ビュー	双方向バインディング	[(target)]='value'

■テンプレート参照変数

・# 変数名
・(change)="0" は、イベントトリガーで値を更新するため必要

```
<input #txtHoge type="text" (change)="0"/>
<div>{{txtHoge.value}}</div>
```

双方向バインディング

・import と @NgModule の imports に FormModule を追加

ルートモジュール (app.modules.ts)

```
import { FormsModule } from '@angular/forms';
   :
@NgModule({
   :
  imports: [
   BrowserModule,
   FormsModule,
     :
```

ビュー

・input/textarea/select などフォーム要素をバインドするには、ngModel を利用する
・このためには、name 属性の指定が必須
・ngModel を [(ngModel)] とする

```
<select name="selAcion" [(ngModel)]="selectedAction">
    <option *ngFor="let item of testActions" value="{{item}}" >{{item}}</option>
</select>
```

コンポーネント

```
export class AccountComponent implements OnInit {
  testActions: string[] = ['','login','logout','check'];
  selectedAction: string = '';
```

■入力値の加工

・上記は、プロパティバインディング、イベントバインディングを組み合わせて、双方向を実現している。
・データバインディング時に値を加工する場合、[(ngModel)] を [ngModel] と (ngModelChagen) に分解
・$event は入力値そのものを表す

```
<input name="hoge" type="text"
  [ngModel] = "hogeName"
  (ngModelChange) = "hogeName=$event.toUpperCase()" />
```

ルーティング

・https://qiita.com/kazukisugita/items/5c683e93168505720e5f

適用

■プロジェクト作成

```
> ng new app --routing
```

--routing オプションを付与せずにプロジェクトを作成した場合

・/src/app/app-routing.module.ts を追加

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

@NgModule({
  imports: [
    RouterModule.forRoot(routes),
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

・/src/app/app.module.ts

```
import { AppRoutingModule } from './app-routing.module';
   :
@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
  ],
  providers: [],
```

・/src/app/app.component.spec.ts

```
import { RouterTestingModule } from '@angular/router/testing';
   :
describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));
```

■コンポーネント作成

```
> ng g component account --routing
```

■ルーティング定義

・app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AccountComponent } from './account/account.component'
```

```
const routes: Routes = [
  {
    path: 'account', component: AccountComponent
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes),
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

■利用

```
<a routerLink="/account">Account</a>
<router-outlet></router-outlet>
```

コンポーネント

ライフサイクル

ライフサイクル	内容
コンポーネント生成	
コンストラクター	
ngOnChanges	@Input 経由で入力値が設定 / 再設定された
ngOnInit	入力値 (@Inputp) が処理された後、コンポーネントの初期化時 (最初の ngOnChanges メソッドの後で一度だけ)
ngDoCheck	状態の変更を検出したとき
ngAfterContentInit	外部コンテンツを初期化した時 (最初の ngDoCheck メソッドの後で一度だけ)
ngAfterContentChecked	外部コンテンツの変更をチェックした時
ngAfterViewInit	現在のコンポーネントと子コンポーネントのビューを生成した時 (最初の ngAfterContentChecked メソッドの後で一度だけ)
ngAfterViewChecked	現在のコンポーネントと子コンポーネントのビューが変更された時
ngOnDestroyed	コンポーネントが破棄される時
コンポーネント破棄	

サービス

・サービスクラスであることの条件は、@Injectable デコレータを付与することのみ。
・@Injectable デコレータは、コンポーネントに対してサービスを引き渡せることを意味する。

登録

・モジュールにサービスを登録する。
・コンポーネントにも登録できる。この場合コンポーネントと子コンポーネントのみで利用できる。

```
import { HogeService } from './hoge.service';

@NgModule({
   :
  providers: [HogeService],
   :
})
```

依存性注入

・方法を宣言するのは、@NgModule/@Component デコレータの providers パラメータ
・サービスを提供するための Provider オブジェクトを登録する

Provider であることの条件は以下のプロパティを持つこと

プロパティ	内容
provide	サービスを注入する際に利用する DI トークン
useXxxxx	サービスの生成方法 例 userClass: XXXX と指定するとクラス XXXX を常に new でインスタンス化する
multi	同一の DI トークンに対して複数の Provider を追加するか

■ useXxxx

プロパティ	内容
useClass	指定されたクラスを注入のたびにインスタンス化
useValue	指定されたオブジェクトを常に引き渡す (同じ値になる)
useExisting	指定されたトークンのエイリアスを生成
useFactory	指定されたファクトリー関数で注入の際にオブジェクトを生成

useClass

・常に新たなインスタンスを生成する

```
providers: [
  { provide: HogeService, useClass: HogeService }
]
```

useValue

・常に同じオブジェクトを注入する
・クラスのインスタンスを渡す

```
providers: [
  { provide: HogeService, useVlaue: new HogeService() }
]
```

useExisting

・トークンのエイリアスを生成

```
providers: [
  { provide: HogeService, useClass: HogeService }、
  { provide: HogeAliasService, useExisting: HogeService }、
]
```

互換性維持など、別のトークンから同一インスタンスを取得したい場合などに利用

useFactory

・ファクトリー関数経由でインスタンスを生成

```
providers: [
  { provide: HogeService, useFactory: () => {
      let service = new HogeService();
      service.foo = "bar";
      return service;
    }
  }
]
```

Modules

Http

・https://angular.io/guide/http

RxJs

コンポーネントからサービスのデータを購読する

・https://angular.io/guide/component-interaction#!#bidirectional-service

Service

```
import { Subject } from 'rxjs/subject';
import { User } from './user';

@Injectable()
export class AccountService {
  private userChangeAnnouncedSource = new Subject<User>();
  userChangeAnnounced$ = this.userChangeAnnouncedSource.asObservable();
    :
  announceUserChange(user: User) {
    this.userChangeAnnouncedSource.next(user);
  }
```

:

Component

```
import { Subscription } from 'rxjs/Subscription';

export class AccountComponent implements OnInit, OnDestroy {
    user: User;
    subscription: Subscription

  ngOnInit() {
    this.subscription = this.accountService.userChangeAnnounced$.subscribe((user:User) => {
      this.user = user;
    });
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
      :
```

UI

Angular Material

- https://material.angular.io/

■レスポンシブレイアウト

- https://material.angular.io/cdk/layout/overview

テスト

ユニットテスト

■ Karma

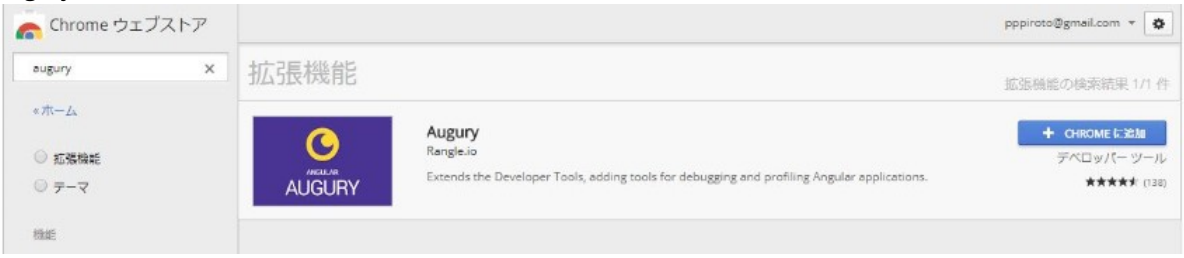
- https://karma-runner.github.io/2.0/index.html
- karma.conf.js

Tips

機能

■ Chrome 拡張機能

augury



■ Facebook SDK

・https://developers.facebook.com/docs/javascript

■ Hello.js 認証をまとめる JavaScript ライブラリ

・http://adodson.com/hello.js/#scope

文法

■ Javascript のグローバルオブジェクトを利用

```
declare const hoge;
```

コード

■コードからルーティング

```
import { Router } from '@angular/router';
  :
constructor(private router: Router) {}
  :
this.router.navigate(['/hoge'])
```

■ CSRF

・https://angular.io/api/common/http/HttpClientXsrfModule
・https://angular.io/guide/http
・Angular + HttpClientXsrfModule + Flask で CSRF

参照

・Angular CLI でサブディレクトリにビルドする方法