环境搭建

- 安装nodejs稳定版本：http://nodejs.cn/
  - 安装cnpm：npm install cnpm -g --registry=https://registry.npm.taobao.org（可不安装）
- 使用npm/cnpm命令安装angular/cli（只需要安装一次）
  - npm install -g @angular/cli 或者 cnpm install -g @angular/cli
- 输入ng v验证是否安装成功

创建项目

- 创建一个文件夹——在文件夹路径中输入cmd
- ng new angulardemo01：创建项目（项目名称）
  - ng new angulardemo02 --skip-install（跳过依赖创建项目）
  - cd angulardemo02后，再npm install安装依赖
- cd angulardemo01后，ng serve --open：运行项目
- vsCode打开，安装angular插件

创建组件

- cd angulardemo01后，ng g component components/news

ts+angular

声明属性

- public 公有*(默认)：可以在这个类里面使用、也可以在类外面使用
- protected 保护类型：只有在当前类和它的子类里面可以访问
- private 私有类型：只有在当前类才可以访问这个属性

绑定数据

```
<div>{{ student }}</div>
```

绑定属性

```
<div title="我是一个div"></div>      //静态
<div [title]="student"></div>       //动态
```

绑定样式

- [ngclass]

```
<div class="red">ngClass演示</div>
<div [ngClass]="{ blue: false, red: true }">ngClass演示</div>
<div [ngClass]="{ blue: flag, red: !flag }">ngClass演示</div>
.blue{
    color: blue;
}
.red{
    color: red;
}
public flag: boolean = true;
```

- [ngstyle]

```
<div [ngStyle]="{ color: 'blue' }">ngStyle演示</div>
<div [ngStyle]="{ color: attr }">ngStyle演示</div>
public attr: string = 'red';
```

绑定html

```
<div [innerHTML]="content"></div>
```

简单运算

```
<div>1+2={{ 1+2 }}</div>
```

引入图片

```
<img src="/assets/images/111.jpg" alt="">   //静态
<img [src]="picUrl" alt="">                          //动态

public picUrl =
    'https://www.baidu.com/img/PCtm_d9c8750bed0b3c7d089fa7d55720d6cf.png';
```

循环数组*ngFor

```
<ul>
  <li *ngFor="let item of arr">
    {{ item }}
  </li>
</ul>
<ul>
  <li *ngFor="let item of list; let key = index">   //显示索引
    {{ key }}---{{ item.title }}
  </li>
</ul>
```

条件判断

- *nglf

```
<div *ngIf="flag">ccc</div>
<div *ngIf="!flag">ttt</div>
<div [hidden]="flag">ccc</div>   //[hidden]隐藏属性

public flag: boolean = false;
```

- *ngswitch

```
<span [ngSwitch]="orderStatus">
  <p *ngSwitchCase="1">已经支付</p>
  <p *ngSwitchCase="2">已经确认</p>
  <p *ngSwitchDefault>未支付</p>
</span>

public orderStatus: number = 1;
```

管道

- http://bbs.itying.com/topic/5bf519657e9f5911d41f2a34

```
<span>{{ today | date: "yyyy-MM-dd mm:ss" }}</span>
public today: any = new Date();
```

事件

```
<button (click)="run()">执行</button>
run() {
   alert('1111');
}
```

```
<button (click)="getDate()">执行</button>
getDate() {
   alert(this.attr);
}
```

```
<button (click)="setDate()">执行</button>
setDate() {
   this.attr="test"
}
```

表单事件/获取事件对象

```
<input type="text" (keydown)="keyDown()" />
keyDown() {  }

<input type="text" (keydown)="keyDown($event)" />
keyDown(e: any) {
    console.log(e.target.value);           //e.target获取dom对象
  }

<button (click)="runEvent($event)">执行</button>
runEvent(event){
        let dom:any = event.target
        dom.style.color = "red"
}
```

双向数据绑定

- app.module.ts

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

- 组件中

  ```
  <input type="text" [(ngModel)]="keywords" />   //[]绑定属性 {}绑定事件
  {{ keywords }}
  <button (click)="changeKey()">改变keywords</button>

  public keywords: string = '默认值';
  ```

- 表单

```
<h2>人员登记系统</h2>
<div class="people_list">
  <ul>
    <li>
      姓名: <input
        class="formInput"
        type="text"
        name=""
        id=""
        [(ngModel)]="peopleInfo.username"
      />
    </li>
    <li>
      性别:
      <input
        type="radio"
        value="1"
        name="sex"
        id="sex1"
        [(ngModel)]="peopleInfo.sex"
      />
      <label for="sex1">男</label>
      <input
        type="radio"
        value="2"
        name="sex"
        id="sex2"
        [(ngModel)]="peopleInfo.sex"
      />
      <label for="sex2">女</label>
    </li>
    <li>
      城市: <select name="city" id="city" [(ngModel)]="peopleInfo.city">
        <option [value]="item" *ngFor="let item of peopleInfo.cityList">
          {{ item }}
        </option>
      </select>
    </li>
    <li>
      爱好: <span *ngFor="let item of peopleInfo.hobby; let key = index">
        <input
          type="checkbox"
          name=""
          [id]="'check' + key"
          [(ngModel)]="item.checked"
        />
        <label [for]="'check' + key">{{ item.title }}</label>
         
      </span>
    </li>
    <li>
      备注: <textarea
        name=""
```

```
        id=""
        cols="30"
        rows="10"
        [(ngModel)]="peopleInfo.mark"
      ></textarea>
    </li>
  </ul>
  {{ peopleInfo | json }}
  <button (click)="doSumit()"></button>
</div>
```

```
public peopleInfo: any = {
  username: '',
  sex: '1',
  cityList: ['北京', '上海', '深圳'],
  city: '北京',
  hobby: [
    { title: '吃饭', checked: true },
    { title: '睡觉', checked: false },
    { title: '打豆豆', checked: false },
  ],
  mark: '',
};
```

服务及storage数据持久化

- 创建服务ng g service service/storage
- app.module.ts引入并声明

```
import { StorageService } from './service/storage.service';
@NgModule({
  providers: [StorageService],
})
```

- 在需要用到的组件中再次引入，初始化，使用持久化数据

```
import { StorageService } from '../../service/storage.service';
export class SearchComponent implements OnInit {
  public keyword: string = '';
  public historyList: any[] = [];
  constructor(public storage: StorageService) {
  }
  ngOnInit() {
    if (this.storage.get('searchList')) {
      this.historyList = this.storage.get('searchList');
    }
  }
  doSearch() {
    if (this.historyList.indexOf(this.keyword) === -1) {
      this.historyList.push(this.keyword);
```

```
      this.storage.set('searchList', this.historyList);
      this.keyword = '';
    }
  }
  delKeyword(index: any) {
    this.historyList.splice(index, 1);
    this.storage.set('searchList', this.historyList);
  }
}
```

- storage.service.ts

```
export class StorageService {
  constructor() {}
  set(key: string, value: any) {
    localStorage.setItem(key, JSON.stringify(value));
  }
  get(key: string) {
    return JSON.parse(localStorage.getItem(key) || '');
  }
  remove(key: string) {
    localStorage.removeItem(key);
  }
}
```

dom操作(viewChild)

- 原生js
  - 组件html文件

```
<div id="box">
    this is box
</div>
<div id="box1" *ngIf="flag">
    this is box
</div>
```

  - 组件ts文件

```
export class HomeComponent implements OnInit {
  public flag: boolean = true;
  constructor() {}
  ngOnInit() {
    let box: any = document.getElementById('box');
    box.style.color = 'red';
  }
  ngAfterViewInit() {
    let box1: any = document.getElementById('box1');
    box1.style.color = 'blue';
  }
}
```

- viewChild获取dom节点/调用子组件方法
  - 组件html文件

```
<app-header #header></app-header>          //子组件
<div #myBox>             //dom节点
    一个newsbox
</div>
```

  - 组件ts文件

```
import { Component, OnInit, ViewChild } from '@angular/core';
export class NewComponent implements OnInit {
  @ViewChild('myBox') myBox: any;
  @ViewChild('header') header: any;
  ngAfterViewInit(){
    this.myBox.nativeElement.style.color="red"
    this.header.run();          //获取自组件中的run方法
  }
}
```

angular生命周期

- constructor：构造函数中除了使用简单的值对局部变量进行初始化之外，什么都不应该做。(非生命周期函数)
- ngOnChanges()：被绑定的输入属性的值发生变化时调用（父子组件传值的时候会触发）
- **ngOnInit()：请求数据操作**
- ngDoCheck()：自定义操作（数据改变就会触发）
- ngAfterContentInit()：组件渲染完成调用
- ngAfterContentChecked()：组件初始化完成做一些自定义操作（数据改变就会触发）
- **ngAfterViewInit()：视图加载完成，做一些dom操作**
- ngAfterViewChecked()：自定义操作（数据改变就会触发）
- **ngOnDestroy()：销毁**

RxJS异步数据编程

- 一种针对异步数据流的编程。它将一切数据，包括HTTP请求、DOM事件或者普通数据等包装成流的形式，然后用强大丰富的操作符对流进行处理，以同步编程的方式处理异步数据，并组合不同的操作符来实现功能。
- 回调函数解决异步

```
// 【request.service.ts中】
// 回调函数解决异步
getCallbackData(cb: any) {
  setTimeout(() => {
    var name = 'test';
    cb(name);
  }, 1000);
}
// 【组件中】
this.request.getCallbackData((data: any) => {
  console.log(data);
});
```

- promise解决异步

```
// 【request.service.ts中】
// 回调函数解决异步
getPromiseData() {
  return new Promise((resolve: any) => {
    setTimeout(() => {
      var name = 'test';
      resolve(name);
    }, 1000);
  });
}
// 【组件中】
this.request.getPromiseData().then((data) => {
  console.log(data);
});
```

- Rxjs解决异步

```
// 【request.service.ts中】
// rxjs解决异步
import { Observable } from 'rxjs';
getRxjsData() {
  return new Observable((observer: any) => {
    setTimeout(() => {
      var name = 'test--rxjs';
      observer.next(name);
      // observer.error('失败');
    }, 3000);
  });
}
// 【组件中】
let stream = this.request.getRxjsData();
let data = stream.subscribe((data) => {
```

```
  console.log(data);
});
setTimeout(() => {
  data.unsubscribe();   //取消订阅
}, 1000);
```

- 多次执行

```
// rxjs解决异步--多次执行
getRxjsIntervalData() {
  let count = 0;
  return new Observable((observer: any) => {
    setInterval(() => {
      count++;
      var name = 'test--rxjs--Interval' + count;
      observer.next(name);
    }, 1000);
  });
}
// 【组件中】
let streamInterval = this.request.getRxjsIntervalData();
streamInterval.subscribe((data) => {
  console.log(data);
});
```

- Rxjs工具函数

```
// 【request.service.ts中】
getRxjsIntervalNum() {
  let count = 0;
  return new Observable((observer: any) => {
    setInterval(() => {
      count++;
      observer.next(count);
    }, 1000);
  });
}
// 【组件中】
import { map, filter } from 'rxjs/operators';
// 用工具方法处理返回的数据
let streamNum = this.request.getRxjsIntervalNum();
// filter
streamNum
  .pipe(
    filter((value: any): any => {
      if (value % 2 == 0) {
        return true;
      }
    })
  )
  .subscribe((data) => {
    console.log(data);
```

```
  });
// map
streamNum
  .pipe(
    map((value: any): any => {
      return value * 2;
    })
  )
  .subscribe((data) => {
    console.log(data);
  });
// filter+map合并使用
streamNum
  .pipe(
    filter((value: any): any => {
      if (value % 2 == 0) {
        return true;
      }
    }),
    map((value: any): any => {
      return value * 2;
    })
  )
  .subscribe((data) => {
    console.log(data);
  });
```

请求数据

- angular自带模块
  - app.module.ts

```
import { HttpClientModule, HttpClientJsonpModule } from '@angular/common/http';
@NgModule({
  imports: [BrowserModule, HttpClientModule, HttpClientJsonpModule],
})
```

  - 组件中

```
import { HttpClient,HttpHeaders } from '@angular/common/http';
export class HomeComponent implements OnInit {
  public list: any[] = [];
  constructor(public http: HttpClient) {}
  // 【get】
  getData() {
    this.http
      .get('http://a.itying.com/api/productlist')
      .subscribe((response: any) => {
        this.list = response.result;
      });
```

```
  }
  // 【post】
  doLogin() {
  const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' }),
  };
  this.http
    .post(
      'http://127.0.0.1:3000/dologin',
      { username: 'www', age: 20 },
      httpOptions
    )
    .subscribe((response) => {
      console.log(response);
    });
  }
  // 【jsonp】
  getJsonp() {
    // 确定服务器是否支持jsonp：url后面加?callback=xxx或者?cb=xxx 与下面第二个参数对应
    this.http
      .jsonp('http://a.itying.com/api/productlist', 'callback')
      .subscribe((response: any) => {
        this.list = response.result;
      });
  }
}
```

- 第三方模块axios（服务中封装）
  - npm install axios --save
  - httpservice.service.ts中

```
import axios from 'axios';
export class HttpserviceService {
  constructor() {}
  axiosGet(api: any) {
    return new Promise((resolve) => {
      axios.get(api).then(function (response) {
        resolve(response);
      });
    });
  }
}
```

  - app.module.ts中

```
import { HttpserviceService } from './service/httpservice.service';
@NgModule({
  providers: [HttpserviceService],
})
```

  - 组件中

```
import { HttpserviceService } from '../../service/httpservice.service';
export class HomeComponent implements OnInit {
  constructor(
    public httpservice: HttpserviceService
  ) {}
  getAxiosData() {
    this.httpservice
      .axiosGet('http://a.itying.com/api/productlist')
      .then((response) => {
        console.log(response);
      });
  }
}
```

路由

- 创建项目时选择带路由
- 创建组件
- app-routing.module.ts中

```
import { HomeComponent } from './components/home/home.component';
import { NewsComponent } from './components/news/news.component';
import { ProductionComponent } from './components/production/production.component';
import { NewsdetailComponent } from './components/newsdetail/newsdetail.component';
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'news', component: NewsComponent },
  { path: 'production', component: ProductionComponent },
  // 【get传值】写法
  { path: 'newsdetail', component: NewsdetailComponent },
  // 【动态路由传值】写法
  { path: 'newsdetail/:aid', component: NewsdetailComponent },
  // 匹配不到时加载的组件 需写在最后
  { path: '**', redirectTo: 'home' },
];
```

- app.component.html中

```
<a [routerLink]="['/home']" routerLinkActive="active">首页</a>
<hr />
<a [routerLink]="['/news']" routerLinkActive="active">新闻</a>
<hr />
<a routerLink="/production" routerLinkActive="active">商品</a>
<router-outlet></router-outlet>
```

- app.component.scss中

```
.active{
    color:red
}
```

- news.component.ts中

```
export class NewsComponent implements OnInit {
  public list: any[] = [];
  constructor() {}
  ngOnInit() {
    for (let i = 0; i < 10; i++) {
      this.list.push('第' + i + '条数据');
    }
  }
}
```

- news.component.html中，跳转传值

```
<!-- get传值写法 -->
<ul>
  <li *ngFor="let item of list; let key = index">
    <a [routerLink]="['/newsdetail']" [queryParams]="{ aid: key }">
        {{ key }}---{{ item }}</a>
  </li>
</ul>
<!-- 动态路由传值写法 -->
<ul>
  <li *ngFor="let item of list; let key = index">
    <a [routerLink]="['/newsdetail', key]">{{ key }}---{{ item }}</a>
  </li>
</ul>
```

- news.component.ts中，接收传值

```
import { ActivatedRoute } from '@angular/router';
export class NewsdetailComponent implements OnInit {
  constructor(public route: ActivatedRoute) {}
  ngOnInit() {
    // get传值写法
    this.route.queryParams.subscribe((data) => {
        console.log(data);
    });
    // 动态路由传值写法
    this.route.params.subscribe((data) => {
      console.log(data);
    });
  }
}
```

动态路由的js跳转

- app-routing.module.ts

```
import { ProductionComponent } from './components/production/production.component';
import { ProductiondetailComponent } from
'./components/productiondetail/productiondetail.component';
const routes: Routes = [
  { path: 'production', component: ProductionComponent },
  { path: 'productiondetail/:pid', component: ProductiondetailComponent },
];
```

- production.component.html

```
<a [routerLink]="[ '/productiondetail', '123' ]">跳转到商品详情</a>
<button (click)="goProductiondetail()">js跳转路由到商品详情</button>
```

- production.component.ts

```
import { Router } from '@angular/router';
export class ProductionComponent implements OnInit {
  constructor(public router: Router) {}
  goProductiondetail() {
    this.router.navigate(['/productiondetail', '123']);
  }
}
```

get传值的js跳转

- app-routing.module.ts

```
import { ProductionComponent } from './components/production/production.component';
import { ProductiondetailComponent } from
'./components/productiondetail/productiondetail.component';
const routes: Routes = [
  { path: 'production', component: ProductionComponent },
  { path: 'productiondetail', component: ProductiondetailComponent },
];
```

- production.component.html

```
<button (click)="goProductiondetail()">get传值跳转路由</button>
```

- production.component.ts

```
import { Router, NavigationExtras } from '@angular/router';
export class ProductionComponent implements OnInit {
  constructor(public router: Router) {}
  goProductiondetail() {
    // 跳转并get传值
    let queryParams: NavigationExtras = {
      queryParams: { aid: 123 },
    };
    this.router.navigate(['/productiondetail'], queryParams);
  }
}
```

父子路由

- app-routing.module.ts

```
const routes: Routes = [
  {
    path: 'home',
    component: HomeComponent,
    children: [
      { path: 'welcome', component: WelcomeComponent },
      { path: 'setting', component: SettingComponent },
      { path: '**', redirectTo: 'welcome' },
    ],
  },
  { path: '**', redirectTo: 'home' },
];
```

- home.component.html

```
<div class="content">
  <div class="left">
    <a [routerLink]="['/home/welcome']">欢迎首页</a>
    <br />
    <a [routerLink]="['/home/setting']">系统设置</a>
  </div>
  <div class="right">
    <router-outlet></router-outlet>
  </div>
</div>
```

项目步骤

- 有几个页面，新建几个组件
- 配置路由
- 编写结构和样式

    - 结构只需要把body中的结构放在html中
    - 基础样式放在style.scss中，组件样式放在对应最贱scss文件中
    - 图片放在assets/images中
- 实现路由跳转逻辑
- 实现请求数据逻辑（封装成服务）