1. Angular 常用知识点总结
    1. 总览
        1. 基础构造块：NgModule
        2. 组件定义视图，组件使用服务
        3. 装饰器
        4. 元数据
        5. 模板
        6. 路由
    2. 知识点1：模块
        1. NgModule元数据
            1. declarations
            2. exports
            3. imports
            4. providers
            5. bootstrap
        2. NgModule与组件
    3. 知识点2：组件
        1. 组件元数据
            1. selector
            2. templateUrl
            3. providers
    4. 知识点3：模板
    5. 知识点4：元数据
    6. 知识点5：数据绑定
        1. 插值
        2. 属性绑定
        3. 事件绑定
        4. 双向数据绑定
    7. 知识点6： 管道
    8. 知识点7：指令
        1. 结构型指令
        2. 属性型指令
    9. 知识点8：服务
    10. 知识点9：依赖注入
    11. 知识点10：路由
    12. 知识点11： SSR（Angular Universal）
        1. 好处
        2. 流程

3. Universal的模板引擎
4. 过滤请求的URL
5. 在服务端使用绝对 URL 进行 HTTP（数据）请求

Angular 常用知识点总结

总览

基础构造块：NgModule

一个为组件提供编译上下文的容器，可以关联组件与服务，作用域由NgModule定义。

可以导入其他模块的功能，和导出功能为其他模块使用。

Angular应用就是由一组NgModule定义出的。应用至少会有一个用于引导应用的根模块（AppModule，位于app.module.ts），和其他特性模块。

```
@NgModule({
  declarations: [
    AppComponent,
    HeroesComponent,
    HeroDetailComponent,
    MessagesComponent,
    DashboardComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

组件定义视图，组件使用服务

组件定义视图的的显示逻辑，组件使用服务提供的与视图不相关的功能如数据源。服务提供者作为依赖，注入到组件中

```
@Component({
  selector: 'app-messages',
  templateUrl: './messages.component.html',
  styleUrls: ['./messages.component.scss']
})
export class MessagesComponent implements OnInit {

  constructor(public messageService: MessageService) { }

  ngOnInit(): void {
  }

}
```

装饰器

模块，组件，服务都是使用装饰器的类，装饰器会标记它们的类型并提供元数据。

模块

```
@NgModule()
```

组件

```
@Component()
```

服务

```
@Injectable ()
```

元数据

将组件类和定义视图的模板相关联。

```
@Component({
selector: 'app-messages',
templateUrl: './messages.component.html',
styleUrls: ['./messages.component.scss']
})
```

模板

把HTML与Angular指令相组合，在渲染HTML之前，修改HTML。

```
<div *ngIf="messageService.messages.length">
  <h2>Message</h2>
  <button class="clear" (click)="messageService.clear()">Clear message</button>
  <div *ngFor="let message of messageService.messages">{{message}}</div>
</div>
```

路由

定义视图间导航路径。

```
const routes: Routes = [
  {path: 'heroes', component: HeroesComponent},
  {path: 'dashboard', component: DashboardComponent},
  {path: 'detail/:id', component: HeroDetailComponent},
  {path: '', redirectTo:'/dashboard', pathMatch:'full'}
];
```

知识点1：模块

为组件提供编译上下文。将组件与服务关联成功能单元。

Angular应用包含根模块(AppModule)作为启动引导。

可以导入和导出功能。

NgModule元数据

NgModule是一个带有 `@NgModule()` 装饰器的类。

`@NgModule()` 是一个函数，它接受一个元数据对象{}。

```
@NgModule({
declarations: [
AppComponent,
```

```
  HeroesComponent,
  HeroDetailComponent,
  MessagesComponent,
  DashboardComponent
  ],
  imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  HttpClientModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
  })
  export class AppModule { }
```

declarations

属于本NgModule的组件，指令，管道。

exports

导出本模块使其可用于其他模块

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

imports

用于本模块组件里的外部导入模块

providers

本模块向全局贡献的服务

bootstrap

主视图，根模块特有

NgModule与组件

NgModule为组件提供编译上下文。

组件与其模板共同定义视图。

知识点2：组件

组件控制小部分视图区域。

在类中定义组件显示逻辑，为视图提供支持。

组件通过由属性和方法组成的API与视图交互。

组件元数据

元数据告诉了Angular去哪里找它所需要的代码块。它把模板与组件相关联。

```
@Component({
selector: 'app-messages',
templateUrl: './messages.component.html',
styleUrls: ['./messages.component.scss'],
providers: [HeroService]
})
```

selector

css选择器,告诉Angular，HTML出现选择器对应标签时，插入该组件实例

```
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
```

templateUrl

该组件HTML模板的地址，定义宿主视图

providers

组件所需服务

知识点3：模板

HTML来告诉Angular如何渲染。

使用数据绑定来协调DOM中数据。

使用管道在显示出来前进行转换。

使用指令来把程序逻辑应用到显示内容上。

知识点4：元数据

模块，组件，服务都是使用装饰器的类，装饰器会标记它们的类型并提供元数据。

元数据告诉了Angular去哪里找它所需要的代码块。它把模板与组件相关联。

```
@Component({
selector: 'app-messages',
templateUrl: './messages.component.html',
styleUrls: ['./messages.component.scss'],
providers: [HeroService]
})
```

知识点5：数据绑定

```
<li>{{hero.name}}</li>
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<li (click)="selectHero(hero)"></li>
```

插值

`{{hero.name}}`

属性绑定

`[hero]`属性绑定把父组件 `HeroListComponent` 的 `selectedHero` 的值传到子组件 `HeroDetailComponent` 的 `hero` 属性中。

事件绑定

当用户点击某个英雄的名字时，(click) 事件绑定会调用组件的 selectHero 方法

双向数据绑定

```
<input type="text" id="hero-name" [(ngModel)]="hero.name">
```

把事件绑定和属性绑定结合。属性绑定让数据从组件流入输入框。用户的修改通过事件绑定流回组件，把属性值设为新值。

知识点6： 管道

在HTML模板中声明，显示值的转换逻辑。操作符为 |

```
<!-- Default format: output 'Jun 15, 2015'-->

 <p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->

<p>The date is {{today | date:'fullDate'}}</p>
```

知识点7：指令

渲染HTML模板时，根据指令对DOM进行转换。常常作为属性出现在元素标签上

结构型指令

增加，删除，替换DOM元素修改布局。

```
<li *ngFor="let hero of heroes"></li>
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

*ngFor是迭代器，遍历数组heroes为每个hero渲染一个li

*ngIf是条件渲染，当selectedHero存在时，渲染HeroDetail组件

属性型指令

修改现有属性的显示值，行为，外观。

```
<input type="text" id="hero-name" [(ngModel)]="hero.name">
```

ngModel设置了，属性值的显示，和对change事件的响应。

知识点8：服务

组件把例如从服务器获取数据的工作委托给服务，让组件只关注用户体验。

通过把处理任务定义到服务类中，它可以被各种组件使用。

服务也可依赖于其他服务，例如这里的MessageService

```
@Injectable({
  providedIn: 'root',
})
export class HeroService {
  private heroesUrl = 'api/heroes';
  httpOptions = {headers: new HttpHeaders({'Content-Type': 'application/json'})}
  constructor(
    private messageService: MessageService,
    private http: HttpClient
  ) {}

  private log(message: string) {
    this.messageService.add(`HeroService: ${message}`);
  }

  getHeroes(): Observable<Hero[]> {
    return this.http
      .get<Hero[]>(this.heroesUrl)
      .pipe(catchError(this.handleError<Hero[]>('getHeroes', [])));
  }

  getHero(id: number): Observable<Hero | undefined> {
    const hero = HEROES.find((h) => h.id === id);
    this.messageService.add(`HeroService: fetched hero id= ${id}`);
    return of(hero);
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.log(error);
```

```
      this.log(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
  updateHero(hero: Hero): Observable<any | undefined> {
    return this.http.put(this.heroesUrl, hero, this.httpOptions)
    .pipe(tap(_=> this.log(`updated hero id=${hero.id}`)),
    catchError(this.handleError<any>('updateHero'))
    )
  }
```

知识点9：依赖注入

依赖注入把应用逻辑分解为各种服务，让这些服务用于各个组件。

把服务注入到组件里，让组件类可以访问服务类。

```
@Injectable({
providedIn: 'root',
})
```

修饰器`@Injectable()`说明该服务可以作为依赖注入

服务可以在元数据中把自己注册为提供者，让自己随处可用。

当在根提供服务时，Angular会为HeroService创建一个共享实例，让他随处可用。

```
constructor(heroService: HeroService)
```

将依赖项注入组件的构造器中

知识点10：路由

当浏览器URL变化时，路由器会查找对应的Route。

通过RouterModule.forRoot()方法配置路由器，把结果添加入imports

```
const routes: Routes = [
  {path: 'heroes', component: HeroesComponent, data: { title: 'Heroes List' }},
```

```
  {path: 'dashboard', component: DashboardComponent},
  {path: 'detail/:id', component: HeroDetailComponent},
  {path: '', redirectTo:'/dashboard', pathMatch:'full'}
];

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

每个Route都会把URLpath映射到一个组件。

空路径表示默认路径

data属性存放与路由相关的数据。例如

```
data: { title: 'Heroes List' }
```

知识点11： SSR（Angular Universal）

标准的Angular运行在浏览器，DOM渲染页面。Angular Universal会在服务端运行，生成静态应用界面，再通过客户端启动。

好处

1. 帮助SEO
2. 提高手机端性能
3. 快速显示第一页

流程

服务器会把客户端对页面的请求传给NgUniversal的ngExpressEngine，调用renderModule()函数。

renderModule()函数，接受HTML模板，组件模块和决定组件显示的路由作为输入

该路由从客户端请求传给服务器，服务器把渲染好的页面作为Promise返回。

Universal的模板引擎

```
server.engine('html', ngExpressEngine({
  bootstrap: AppServerModule,
}));
```

ngExpressEngine是对renderModule()的封装，它把客户端请求转化成服务器渲染的HTML界面。

过滤请求的URL

服务器将对页面的请求和其他请求分开。

应用路由的请求不带扩展名

```
// All regular routes use the Universal engine
server.get('*', (req, res) => {
  res.render(indexHtml, { req, providers: [{ provide: APP_BASE_HREF, useValue:
req.baseUrl }] });
});
```

静态页面的请求会带有扩展名。

```
// Serve static files from /browser
server.get('*.*', express.static(distFolder, {
  maxAge: '1y'
}));
```

数据请求会带有/api开头

```
// TODO: implement data requests securely
server.get('/api/**', (req, res) => {
  res.status(404).send('data requests are not yet supported');
});
```

在服务端使用绝对 URL 进行 HTTP（数据）请求

在服务端渲染的应用中，HTTP URL 必须是绝对的 （例如，`https://my-server.com/api/heroes` ）。

在浏览器中运行时，它们是相对 URL。