Лекция 24. Angular. Routing, Композиция компонентов

[**Angular. Композиция компонентов**](#_fgtikvh14iz6) **3**

[Взаимодействие между компонентами](#_2yycbei1p7r9) 3

[Двусторонняя связь данных между компонентами](#_1xipm56bizxb) 3

[Привязка к событиям дочернего компонента](#_daa9xfq9q2rm) 4

[**Angular. Router**](#_8h3vpd15nr07) **6**

[Базовая маршрутизация](#_i051o3yihdl4) 6

[URL Matching и порядок маршрутов](#_ucxe7s5sihd1) 8

[Переадресация](#_chltwxbhjtjf) 8

[Создание ссылок](#_w2fp41lhcrg7) 9

[Стилизация активных ссылок](#_m1hb3rhrcowe) 9

[Параметризованные ссылки](#_95ojn84v2ikf) 9

[**Задание для закрепления материала**](#_jniz0ijm4tmx) **11**

# Angular. Композиция компонентов

## Взаимодействие между компонентами

При инициализации нового компонента имеется возможность передать данные во вложенный компонент:

<child-comp [userName]="name" [userAge]="age"></child-comp>

при этом, в дочернем компоненте необходимо объявить привязку к входящим параметрам:

@Input() userName: string;

@Input() userAge: number;

Само собой, декоратор Input должен быть импортирован в секции импорта из модуля @angular/core. В случае, если переменная была изменена в родительском компоненте, ее значение будет изменено и в дочернем.

## Двусторонняя связь данных между компонентами

@Component({

selector: 'app-root',

template: '<inner [(title)]="title"></inner>',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title:string = 'app';

setTitle(title:string) {

this.title = title;

}

}

@Component({

selector: 'inner',

templateUrl: '<input type="text" [ngModel]="title" (ngModelChange)="onChange($event)" />',

styleUrls: ['./inner.component.css']

})

export class InnerComponent implements OnInit{

@Input() title:string;

@Output() titleChange = new EventEmitter<string>();  
 **// Обратите внимание на название переменной Output: название переменной + ключевое слово Change. Это договоренности команды Angular, без следования этим договоренностям связь работать не будет!**

constructor() { }

onChange(title) {

this.title = title;

this.titleChange.emit(this.title);

}

}

Таким образом, при изменении переменной title из родительского компонента, значение будет дублироваться в дочернем и наоборот.

## Привязка к событиям дочернего компонента

Иногда возникает необходимость обработать событие, произошедшее в дочернем компоненте, из родительского:

// Родительский компонент  
@Component({

selector: 'app-root',

template: '<inner (onChanged)="onChanged($event)"></inner>',

styleUrls: ['./app.component.css']

})

export class AppComponent {

onChanged() {

console.log('Fire event in ParentComponent', this)

}

}

// дочерний компонент

@Component({

selector: 'inner',

template: '<button (click)="change('data to pass')">fire parent event</button>',

styleUrls: ['./inner.component.css']

})

export class InnerComponent{

@Output() onChanged = new EventEmitter();

change(item:any) {

this.onChanged.emit(item);

}

}

# Angular. Router

Маршрутизация позволяет сопоставлять запросы к приложению с определенными ресурсами внутри приложения. Это особенно полезно при разработке SPA (Single Page Application)

Ключевым для работы маршрутизации является модуль RouterModule, который располагается в пакете @angular/router. Поэтому при работе с маршрутизацией этот пакет должен быть указан в списке зависимостей в файле package.json.

## Базовая маршрутизация

Для работы с маршрутизацией в первую очередь стоит определить базовый адрес приложения. Для этого возьмем веб-страницу index.html и добавим в секцию <head> элемент <base>:

<head>

<meta charset="utf-8" />

<base href="/" />

<title>Hello Angular 5</title>

</head>

Каждый маршрут сопоставляется с определенным компонентом. Поэтому добавим в проект ряд компонентов. Так, добавим в папку src/app новый файл home.component.ts:

import { Component} from '@angular/core';

@Component({

selector: 'home-app',

template: `<h3>Главная</h3>`

})

export class HomeComponent { }

Далее добавим в папку src/app новый файл about.component.ts:

import { Component} from '@angular/core';

@Component({

selector: 'about-app',

template: `<h3>О сайте</h3>`

})

export class AboutComponent { }

И также добавим еще один файл not-found.component.ts:

import { Component} from '@angular/core';

@Component({

selector: 'not-found-app',

template: `<h3>Страница не найдена</h3>`

})

export class NotFoundComponent { }

Для того, чтобы сопоставить новые компоненты с путями, нам необходимо:  
добавить зависимости в AppModule:

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

Подключить новые страницы:  
import { AppComponent } from './app.component';

import { AboutComponent } from './about.component';

import { HomeComponent } from './home.component';

import { NotFoundComponent } from './not-found.component';

Определить когда какую страницу следует выводить:  
const appRoutes: Routes =[

{ path: '', component: HomeComponent},

{ path: 'about', component: AboutComponent},

{ path: '\*\*', component: NotFoundComponent }

];

и применить маршруты к нашему приложению:

@NgModule({

imports: [ BrowserModule, **RouterModule.forRoot(appRoutes)**],

declarations: [   
 AppComponent,   
 HomeComponent,   
 AboutComponent,   
 NotFoundComponent

],

bootstrap: [ AppComponent ]

})

export class AppModule { }  
  
  
  
Последним штрихом необходимо указать приложению в каком месте будут отображаться наши страницы:

import { Component} from '@angular/core';

@Component({

selector: 'my-app',

template: `<div>

**<router-outlet></router-outlet>**

</div>`

})

export class AppComponent {}

Для использования HTML5 History API можно указать в настройках webpack секцию:

devServer: {

historyApiFallback: true,

}

## URL Matching и порядок маршрутов

При указании списка страниц необходимо соблюдать порядок следования, т.к. модуль роутера выведет страницу, которая первой совпадет с текущим адресом:

const appRoutes: Routes =[

{ path: '\*\*', component: NotFoundComponent }, // любые символы

{ path: '', component: HomeComponent},

{ path: 'about', component: AboutComponent}

];

В этом случае все запросы будут отправляться на страницу NotFoundComponent, т.к. она описана первой и подходит под любой адрес.

## Переадресация

const appRoutes: Routes =[

{ path: '', component: HomeComponent},

{ path: 'about', component: AboutComponent},

{ path: 'contact', redirectTo: '/about', pathMatch:'full'},

{ path: '\*\*', redirectTo: '/'}

];

В данном случае, если страница не найдена, то переход будет произведен на страницу Home, за это отвечает строка   
{ path: '\*\*', redirectTo: '/'}  
  
Еще один новый элемент:

{ path: 'contact', redirectTo: '/about', pathMatch:'full'},   
говорит о том, что при полном совпадении адреса с “contact”, т.е. /contact, пользователь будет перенаправлен на страницу about.

Параметр pathMatch:'full' говорит о том, что совпадение должно быть полным, т.е. адрес /contact/map уже не подходит под описание.

## Создание ссылок

Создание ссылок в шаблоне выглядит достаточно просто:  
template: `<div>

<nav>

<a routerLink="">Главная</a>

<a routerLink="/about">О сайте</a>

</nav>

<router-outlet></router-outlet>

</div>`

Т.е. для создания ссылки, необходимо указать атрибут routerLink в ссылке, который будет вести на конкретный адрес страницы.

## Стилизация активных ссылок

<nav>

<a routerLink="" routerLinkActive="active"

[routerLinkActiveOptions]="{exact:true}">Главная</a>

<a routerLink="/about" routerLinkActive="active"   
 [routerLinkActiveOptions]="{exact:true}">О сайте</a>

</nav>

Атрибут routerLinkActive содержит в себе класс, который применится к ссылке, в случае полного совпадения с адресом страницы.  
 [routerLinkActiveOptions]="{exact:true}" - говорит нам о том, что класс должен применяться только при полном совпадении

## Параметризованные ссылки

Довольно часто требуется передать какой-то параметр через строку адреса, например id товара, страницу которого нужно отобразить. Ссылка в этом случае будет выглядеть следующим образом:  
  
const appRoutes: Routes =[

{ path: '', component: HomeComponent},

{ path: 'about', component: AboutComponent},

**{ path: 'item/:id', component: ItemComponent},**

Обработка же такого параметра в компоненте будет выглядеть следующим образом:

import { Component} from '@angular/core';

import { ActivatedRoute} from '@angular/router';

@Component({

selector: 'item-info',

template: `<h3>Модель {{id}}</h3>`

})

export class ItemComponent {

id: number;

**constructor(private activateRoute: ActivatedRoute){**

**this.id = activateRoute.snapshot.params['id'];**

**}**

}

Параметры адреса страницы можно получить из сервиса ActivatedRoute.

В случае передачи параметров, ссылка на страницу будет выглядеть следующим образом:

<a [routerLink]="['item', '5']">item 5</a>

# Задание для закрепления материала

Доработать поля для игры таким образом, чтобы они реагировали на нажатие:

1. в зависимости от содержимого ячейки(с кораблем, пустая) изменялось отображение ячейки при активации ее противником.

При попадании по кораблю, у корабля должно уменьшаться количество здоровья (целых палуб).

При полном уничтожении корабля он должен пропадать из списка кораблей (справа от игровых полей)

Реализовать механизм хода компьютерного противника.