Angular

概述

- Angular.js 1.x

  2009 年诞生，被 Google 收购，官网：https://angularjs.org/

- Angular 2.x

  2015 年改变，几乎推倒了1.x的所有内容，可以认为是一个新的框架

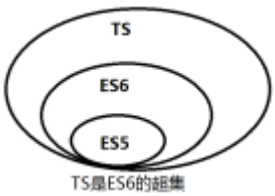
  且语法不再是 JS，而是 TS - TypeScript(微软升级版 JS)

- Mozilla - Firefox　W3C ECMA

- TypeScript : JavaScript 的超集

  官网：https://angular.io　　// 6.x

  中文版： https://angular.cn



One Framework. Mobile & desktop. 一套框架，多种平台移动端 & 桌面端

> DEVELOP ACROSS ALL PLATFORMS　　跨平台
>
> SPEED & PERFORMANCE　　速度与性能
>
> INCREDIBLE TOOLING　　美妙的工具
>
> LOVED BY MILLIONS　　百万粉丝热捧

Angular是一个渐进式的MV*框架，适用于数据操作为主的WEB、移动WEB、桌面应用

Angular的核心概念：

(1) 模块：NgModule，抽象概念，用于封装组件、指令、管道等对象，与项目中的"功能模块"的概念对应。

(2) 组件：Component，是一块可复用的页面区域，包含独立的模板、样式、数据。创建组件的步骤：

1)创建组件　cc.ts

```
@Component({selector: 'cc', template: ''})
export  class  CartCounter{ }
```

2)注册组件　app.module.ts

```
@NgModule({
    declarations: [CartCounter]
```

})

export class AppModule{ }

3)使用组件 app.component.ts/html

<cc>

(3) 数据绑定

1)组件->DOM： {{ msg }}

2)组件->DOM(属性绑定)：

3)DOM->组件(事件绑定)：

4)组件<->DOM(双向绑定)：

(4) 指令

1) 组件

2) 结构型指令：影响DOM结构，必须以*开头

*ngIf、*ngFor*、ngSwitchCase、*ngSwitchDefault

3) 属性型指令：只影响元素的属性，必须用[]括起来

[ngClass]、[ngStyle]、[ngSwitch]、[(ngModel)]

(5) 过滤器/管道

(6) 装饰器和元数据

(7) 服务和依赖注入

(8) 路由

CLI 的安装使用

(1) 下载 Angular-CLI 命令行工具

`npm i -g @angular/cli`

(2) 运行命令行工具，创建一个脚手架项目

`ng new myproject`

(3) 进入项目，运行该项目

`npm run start`

(4) 客户端浏览器访问该项目

http://localhost:4200/

ES6 模块导入和导出

- Vue 中

```
Vue
// Login.js
export default {  }  // 导出，模块的默认导出对象
// main.js
import MyLogin from './Login.js'  // 导入其他模块的默认导出对象，可以随意命名
```

- Angular 中

```
Angular
// Login.js
export var age = 20;
export var isMarried = true;
export function work(){}
export var Emp = { ... }
...
// main.js
import { age, isMarried, emp, ... } from './Login.js'
      // 必须用 {}，变量名必须与模块中声明的一致
```

NG 核心概念

(1) 模块：NgModule

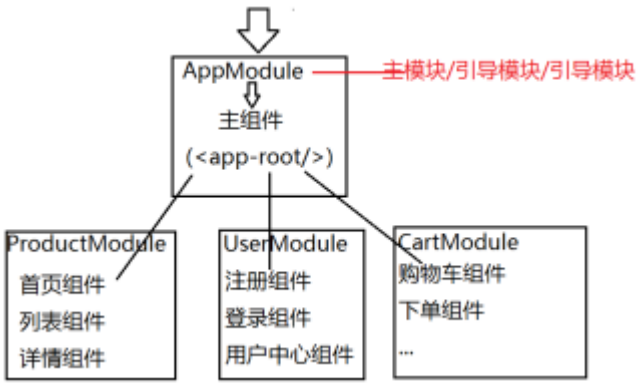
—— Vue.js 中无模块概念

- 与 ES 和 Node.js 中的 “模块”概念

  （一个文件就是一个模块，可以导出内部成员，引入其他模块成员）不同

- NG 中模块指的是项目中的功能模块，如商品模块，用户模块，购物车模块 ...，

  其中包含若干组件、组件等对象



(2) 组件 : Component

- 组件是一块可重复使用的页面区域，有专有的样式、模版、数据等；表现上就是一个自定义的标签

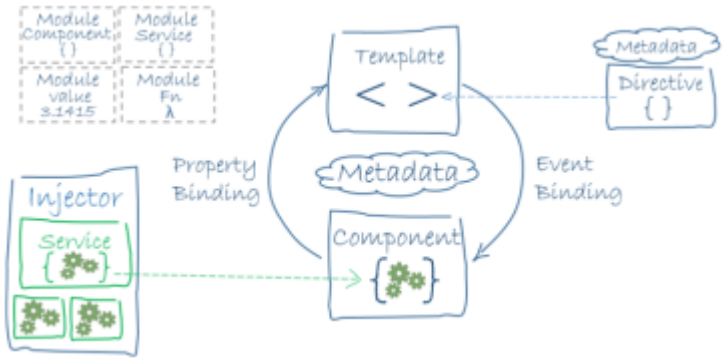
(3) 模块 : Module

(4) 指令 : Directive

(5) 数据绑定 : ngModel

(6) 服务于依赖注入 : DI

(7) 路由 : Route



创建自定义组件

(1) 创建自定义组件

```
//装饰器(Decorator)
@Component({  //元数据(描述数据的数据)对象
  selector:'my-login',
  template:'<div>...</div>'
})
export  class Login{
  uname = 'dingding';    //对象属性：Model数据
  doLogin(){  }         //对象方法：Model方法
}
```

(2) **注册自定义组件（在某个模块中，如AppModule：app.module.ts）**

```
@NgModule({
  declarations: [ Login ]
})
```

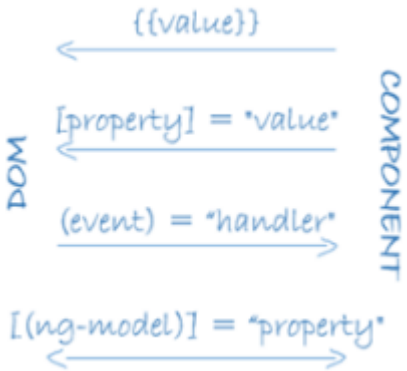
(3) **使用自定义组件**

（自定义组件不能直接用于index.html，只能在主组件中使用 即AppComponent：app.component.html）

```
<my-login></my-login>
```

数据绑定

四种情形：



(1) Model 到 View || 组件 -> DOM：插值表达式 {{ }}

(2) Model 到 View || 组件 -> DOM：属性绑定 [] 不再是 :

(3) View 到 Model || DOM -> 组件：事件绑定 () 不再是 @，无事件修饰符

(4) 双向绑定 || 组件 <-> DOM：ngModel [(ngModel)]

注意：{{ }} 插值表达式中与Vue的不同点：

1) 不能使用 new 关键字，如 new Date()；

2) 不能使用全局函数，如 parseInt()

3) 不能使用全局对象，如 window/JSON

预定义指令

Vue 中的指令：

> v-for / v-if / v-else-if / v-else / v-show / v-bind
>
> v-on / v-html / v-text / v-cloak / v-once / v-pre / v-model

- Angular 中的指令分为三类
  - **组件**：

    NG 中组件继承自指令，是一种特殊的指令

    `Component extends Directive`
  - **结构型指令**：

    会影响模版最终的 DOM 树结构的指令，必须以 `*` 开头

    `*ngFor / *ngIf / *ngSwitchCase / *ngSwitchDefault`
  - **属性型指令**：

会影响当前 DOM 元素的属性的指令，必须用 `[]` 括起来

`[ngClass] / [ngStyle] / [ngSwitch] / [(ngModel)]`

- **ngFor：对数组或对象进行遍历**

```
<li *ngFor="let n of myArr">{{n}}</li>
<li *ngFor="let n of myArr; let i=index;">{{i}}:{{n}}</li>
```

- **ngIf：赋值为 true 则挂载指定元素，false则删除指定元素**

```
<any  *ngIf="变量"></any>
-----------------------------------------------
<any  *ngIf="变量;  else  myelseblock"></any>
<ng-template  #myelseblock >
  <any>...</any>
</ng-template>
```

- **ngSwitch / ngSwitchCase / ngSwitchDefault： 实现多条件判定**

  注意：这三个指令的类型不一样

```
<div [ngSwitch]="orderStatus">
  <p *ngSwitchCase="10">等待付款...</p>
  <div *ngSwitchCase="20">备货中...</div>
  <span *ngSwitchCase="30">运输中...</span>
  <b *ngSwitchCase="40">派送中...</b>
  <i *ngSwitchCase="50">已完成...</i>
  <div *ngSwitchDefault>不可识别的订单状态！</div>
</div>
```

- **ngStyle和ngClass：可以绑定为 String 或 Object**

- **ngModel**

  1) 在主模块中导入FormsModule模块，否则会提示ngModel未知

```
import { FormsModule } from '@angular/forms';
...
@NgModule({ //app.module.ts
  imports:  [ FormsModule ]
})
```

  2) 在组件模块中进行双向数据绑定

```
<input  [(ngModel)="模型变量"]
```

> **如何使用 JS 监视模型数据的改变：**

```
//Vue.js:   使用watch属性
{  data:{kw:''},  watch:{ kw(){  }  }}
-------------------------
//Angular :  使用set访问器
exports class  MyDemo{
  _kw = '';
  get kw(){  return this._kw; }
  set kw(val){   this._kw = val; }
}
```

自定义指令

(1) 创建指令

```
@Direrective({selector: '[xzFocus]'})
export class XzFoucs{
  constructor( el: ElementRef ){
      el.nativeElement  //就是原生DOM对象
  }
}
```

(2) 注册指令

```
@NgModule({
  declarations:[ XzFocus]
})
```

(3) 使用指令

```
<input xzFocus >
```

预定义管道 / 过滤器

Vue.js 中没有预定义过滤器 Vue.js 自定义过滤器： Vue.filter('sex', function(val){   return ....; }) {{ tomSex | sex }}
Angular 提供了预定义过滤器； 也允许用户自定义过滤器；
Angular.js 1.x中，过滤器写作： filter Angular 2.x中，过滤器写作： pipe

(1) date：把对象或数字转换为特定格式的日期字符串

```
{{pubTime | date:'yyyy-MM-dd HH:mm:ss' }}
```

(2) number：把数字显示为指定整数位和小数位的字符串

```
{{score | number:'8.1-3'}}
```

(3) currency：把数字显示为货币形式，逗号+两位小数+货币符号

```
{{productPrice | currency:'￥' }}
```

(4) lowercase：把字符串转换为纯小写形式

(5) uppercase：把字符串转换为纯大写形式

(6) titlecase：把字符串转换为每个单词首字母大写形式

(7) slice：输出数组或字符串中一部分

```
{{msg | slice:2:8}}
```

(8) json：把对象转换为 JSON 字符串格式

自定义管道

(1) 创建管道

```
@Pipe({ name:'sex' })
export class SexPipe{
  transform(val){return ...; }
}
```

(2) 注册管道

```
@NgModule({ declarations: [SexPipe] })
```

(3) 使用管道

```
{{teacherSex | sex }}
```

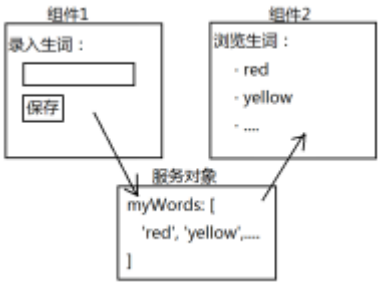
服务 Service

Component

负责维护DOM，并提供DOM操作需要的方法

Service

负责为组件提供底层支持，如日志服务、计时服务、存储服务等



(1) 创建服务对象

```
@Injectable( )
export class WordService{  }
```

(2) 注册服务对象

```
import { WordService } from './word.service'
@NgModule({
  providers: [WordService]
})
```

(3) 使用服务对象

—— 不允许服务直接操作DOM视图，必须注入给组件

```
class XxComponent{
  ws:WordService= null;          // 组件依赖于服务对象
  constructor( s: WordService){   // 将服务注入给组件
    this.ws = s;
  }
}
```

> 重要理论：**依赖注入(Dependency Injection，DI)**

> Dependency：依赖，如果创建 A对象必须现有 B对象，就说 A 依赖于 B，如”Car 依赖于 Moto“。
>
> 依赖对象如何解决：
>
> (1) **穷人式思维：手工创建每一个依赖对象**

```
functioin carRun(){
  var  c1 = new Car( new Moto(), new Sofa( new Skin(),new Thread()... ) );
  c1.run( );
}
```

> (2) **富人式思维：**
>
> 只需要声明自己依赖的对象类型，其所在环境会自动创建需要的对象，并传递进来(注入Inject)

```
function carRun(c2:Car){
  c2.run();
}
```

软件工程：软件设计原则

(1) KISS：Keep It Simple & Stupid

代码越简单越傻瓜越好

(2) DRY：Don't Repeat Yourself

不要重复自己写过的代码，尽量复用代码

(3) SRP：Single Responsibility Principle

单一责任原则

(4) OCP：Open Close Principle

开闭原则，对外部扩展持开放态度，对内部修改持闭合态度

(5) HCLC：High Cohesion，Low Coupling

高内聚低耦合原则

(6) LoD：Law of Demeter

迪米特法则，也称为“最少知识原则”，每个对象知道的数据/操作越少越好

TypeScript

ES 是 ECMA 委员会推出的 JS 行业标准语言；

TS 是 Microsoft 推出的兼容全部 JS 特性的新语言，是 ES 的 **超集。**

> 注意：浏览器只能执行 ES，所有的 TS 必须转换为 ES 语法才能被执行 —— 称为编译。
>
> Angular 脚手架项目自带 TS 编译器，会在服务器端把所有的 TS 代码编译为 ES，供客户端下载使用。

- TS 比 ES 增加的三项主要内容：
  - **(1) 强制类型声明**

    ES是弱类型语言，TS是强类型语言

    —— 所有的属性、形参、函数都需要声明数据类型，

    如string、number、boolean、object、any、xx[ ]
  - **(2) 访问控制修饰符**

    private：私有成员，只能在当前class内部使用

    protected：被保护成员，只能在当前class及子class内部使用

    public：共有成员，可以被其它class直接使用

    提示：一种特殊的简写方法

```
class  MyClass {
  private  age: number = null;
  constructor(age:number){
    this.age = age;
  }
}
=== //等价于
class  MyClass {
  constructor(private  age:number){}
}
```

  - **(3) interface 接口**

    class：类，是对象的模板，可以实例化出多个对象；每个class中可以包含多个 成员属性 + 成员方法。

    interface：接口，是一种特殊的“class”，不能实例化；不能包含成员属性；

    接口中的方法不能有方法体；接口不能被class继承

    —— 接口只能被类“实现(implements)”

    —— 类实现接口时必须实现接口能声明的所有未实现的方法。

    结论：接口用于规范要求类必须提供特定的方法！
  - **(4) 静态成员**

    实例属性：应该无具体值，通过对象应用访问，每个实例都有一份

    静态属性：应该有具体值，通过类名加以访问，所有实例共用一份

```
class Emp{
  ename:string = null;                  //实例属性
  static companyName:string= 'duyi';    //静态属性
}
Emp.companyName= 'duyi';
console.log(Emp.companyName);

var e2 = new Emp();
e2.ename ='Tom';
console.log(e2.ename);
```

生命周期钩子函数

```
@Component({...})
export class MyHeader implements OnInit{
  ngOnInit(){ }          //每个钩子函数都有一个对应的接口
}
```

代码中的 `OnInit` 接口要求 `MyHeader` 必须提供 `ngOnInit()` **生命周期钩子函数。**

组件中通信

Vue.js 中的组件通信

(1)父子间通信

```
// 1) Props Down, Events Up
      Parent: {
          template: '...<child :child-name="myName"/>...'
          data(){return { myName: '苍茫大地' }}
      }
      Child:{
          props: ['childName']
      }
     ----------------------------------------------------------------
      Parent: {
          template: '...<child @childevent="doEvent"/>...'
          methods: {  doEvent(data){ data就是子组件传递的数据 } }
      }
      Child:{
          data(){return { userInput: '' }}
          ...
             this.$emit('childevent',  this.userInput);
      }
// 2)\$refs 和 \$parent
```

(2)兄弟间通信

1) bus 机制

angular 组件间通信

> Angular中兄弟组件间通信一般使用 **Service** 对象；
>
> 父子间通信采用“Props Down，Events Up”机制

> **父 => 子**

```
// 父
@Component({
  template: `<child [childName]="myName"></p> />`;
})
class Parent{
  myName: '苍茫大地';
}
-------------------
// 子
import { Input } from '@angular/core'
@Component({
  template: `<p>{{ childName }}</p>`;
})
class Child{
  @Input  //该装饰器说明这里是一个“输入型属性”
  ChildName = null;
}
```

> **子 => 父**

```
@Component({
    template: '<child (onChangeMyName)="doChange($event)">..' })
class Parent{   doChange(data){  data就是子组件传递的数据 } }
  ----------------------------------------------------------------
@Component({  })
class Child{  userInput = '';
  @Output( )
   onChangeMyName = new EventEmitter( ); //事件发射器
    ...
   this.onChangeMyName.emit( this.userInput );//发射事件
}
```

自定义模块

NgModule：模块，

与ES/Node中的模块不同，不是一个文件，

用于封装组件、指令、管道、服务等对象。

NG中模块的概念与项目中的“功能模块”

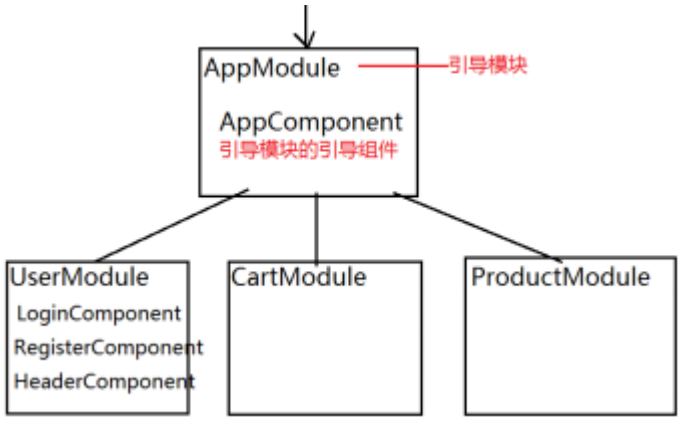
——对应，例如学子商城：

ProductModule——商品模块：首页展示、商品列表、商品详情

UserModule——用户模块：注册、登录、收藏夹、维护注册信息、密码

CartModule——购物车模块：添加、修改、下单

目前，每个项目中都有且只有一个引导启动模块：AppModule（app.module.ts）



如何自定义模块：

```
@NgModule({
  declarations:[ ],   // 声明内部组件、指令、管道
  providers:[ ],      // 声明内部服务对象
  imports: [],        // 声明当前模块使用到了哪些其它模块
  exports: [],        // 声明当前模块内部的哪些对象导出给外界
  bootstrap:[ ]       // 声明当前模块内部的引导组件(只有引导模块有)
                     // —— 只有根模块的引导组件可以用于index.html
})
export class UserModule{  }
```

创建自定义模块： ng g module user

在自定义模块中创建组件： ng g component user/login

提示：自定义模块中创建的组件如果希望被其它模块使用，必须做到：

(1) 在当前模块中声明 declarations

(2) 在当前模块中导出 exports

(3) 在其它模块中导入当前模块 imports

面试题

```
functionf1(){
   var result = f2( );
   console.log(result);    //undefined
}
functionf2(){
  setTimeout(function(){
      return 999;
  }, 0)  //指定的时间间隔到后，把回调函数放到下一次事件循环队列开头
         //  —JS解释器执行完当前队列中所有的任务后再去执行该任务
  console.log('f2执行完成');
}
```

结论：　若 f1 调用 f2；

但 f2 的返回值是在异步方法中返回的，

f1 无法获取到该返回值。

若就希望在 f1 中输出 f2 的异步返回值，解决方案有三种：

(1) f1 提供回调函数给 f2

—— 会产生"回调地狱"

```
function f1(){
  f2( function(data){ console.log(data)} );
}
function f2( fn ){
  setTimeout(function(){
     fn( 999 );
  }, 0)
}
```

(2) Promise

```
function f1(){
  var p = f2();
  p.then( fn ).catch( fn );
}
function f2(  ){
 return new Promise( f1, f2 ){
     setTimeout(function(){
         if()f1(999);
         else f2(888);
     }, 0)
 }
}
```

(3)Observable —— Angular 解决返回异步数据的方案

```
 function f1(){
  var obs = f2();
  obs.subscribe((data)=>{   //订阅“可被关注的对象”
       console.log(data);
  })
}
function f2(  ){
 return new Observable( f1 ){ //返回一个“可被关注/订阅”对象
     setTimeout(function(){
         f1(999)
     }, 0)
 }
}
```

> Vue.js中没有异步访问的问题：
>
> ```
> var ProductList = {
>   data(){return { list: [ ] }},
>   mounted(){
>      axios.get(url).then((res)=>{
>          this.list = res.data;  //数据访问和数据绑定都放在组件中
>      })
>   }
> }
> ```

Angular中实现异步访问

作用：用于异步请求服务器端数据

> 提示：根据Angular的设计原则，数据访问应该放在Service中，
>
> 数据绑定呈现应该放在Component中
>
> —— Angular采用的Observable解决方案。

> H5标准没有Observable技术，它是由第三方工具Rx.js提供的。
>
> https://cn.rx.js.org/

Angular 中实现异步XHR请求的方法：

(1) 在主模块中引入 `HttpClientModule`

```
@NgModule({
  imports: [ HttpClientModule ]
})
```

(2) 创建数据库访问Service对象

```
class ProductService{
  constructor( private http: HttpClient){ }
  getProductList(){
    var observable = this.http.get( 'http://....' )
    return observable;
  }
}
```

(3) 创建需要数据绑定的Component对象

```
class  ProductListComponent{
  list = null;
  constructor(private s: ProductService){}
  onOnInit(){
    this.s.getProductList().subscribe((data)=>{}, (err)=>{})
  }
}
```

Angular 中 SPA 和 Router

> Vue.js 中 VueRouter 实现单页应用的步骤
>
> (1) 视图声明路由插槽：`<router-view></router-view>`
>
> (2) 配置路由词典
>
> ```
> var routes = [
>     {path: '/login', component: LoginComponent}
>     {path: '/*', component: PageNotFoundComponent}
> ]
> ```
>
> (3) 创建路由器对象
>
> ```
> var router = new VueRouter({ routes })
> ```
>
> (4) 在根组件中使用路由器对象
>
> ```
> new Vue({
>    router
> })
> ```
>
> Vue.js中的路由跳转：
>
> (1) HTML中： `<router-link to="/login">跳转</router-link>`
>
> (2) JS 中： `this.$router.push('/login')/back()/forward()....`

Angular 中 SPA 和 Router

> 使用 Angular 官方提供的 RouterModule(@angular/router)
>
> 可以实现Single Page Application (单页应用)

使用步骤：

(1) 在根组件中声明路由插槽/出口

```
<router-outlet></router-outlet>
```

(2) 在根模块中配置路由词典

```
var routes = [
  {path:'index',  component: IndexComponent },
  {path: '**',  component: PageNotFoundComponent }
]
```

(3) 在根模块中使用路由词典

```
imports: [
  RouterModule.forRoot(routes)
]
```

(4) 客户端访问测试

http://127.0.0.1:4200/index

> 注意：
>
> 1) Angular 中的路由地址不能以 `/` 开头
>
> 2) 匹配任意路由地址用 `**` ，而不是 `/*`
>
> 3) 客户端访问时不能在 URL 中出现 index.html，
>
> 也无需书写 #/xxx
>
> —— 最新版本NG利用H5 history.pushState工具实现了此特性
>
> 4) 可以在一个路由组件中使用嵌套路由
>
> 5) 在不同的路由视图间跳转有两种方式：
>
> 方式1：在 HTML 中
>
> `<a routerLink="/login">跳转</a>`
>
> 方式2：在 JS 中
>
> `this.router.navigateByUrl('/login');`

`this.location.back()/forward()`

6) 路由地址中可以携带路由参数(Route Parameter)

1) 声明路由地址时需要声明路由参数

```
{path:'pdetail/:lid', component: ....}
```

2) 路由跳转时需要提供路由参数值

http://127.0.0.1:4200/pdetail/17

3) 组件中通过"当前激活路由"对象读取路由参数

```
ngOnInit(){
  this.activatedRoute.params.subscribe((data)=>{
  // data 就是路由参数数据：{ lid: "17"}
  })
}
```

使用HttpClient发起异步POST请求

经典高阶错误：

```
Failed to load http://127.0.0.1/xz_v1/data/user/register.php: Request header field  Content-Type is not allowed by Access-Control-Allow-Headers *in preflight response*.
register.component.ts:23
```

用户注册异步请求失败

解释说明：

客户端发了预取请求，而服务器给的预取响应中没有"客户端试探性提交的"内容。

解决方法：

(1)要么服务器给出正确的预取响应

(2)要么客户端别发预取请求，直接发正式的请求即可

预取请求(preflight request)和预取响应(preflight response)：

指在客户端与服务器进行正式的请求和响应之前，先进行的"试探性"的请求和响应。

一个典型的预取请求：

OPTIONS /xz_v1/data/user/register.php HTTP/1.1

Access-Control-Request-Method: POST

Access-Control-Request-Headers: content-type

一个典型的预取预取响应：

HTTP/1.1 100 Continue

Access-Control-Allow-Methods: [get, post, put, delete]

Access-Control-Allow-Headers: [content-type, cache-control]

跨域问题解决方案CORS相关的响应消息头部：

Access-Control-Allow-Origin:

　　服务器允许哪些客户端发起请求，如果客户端携带了身份认证信息，此处的值不允许使用 `*`

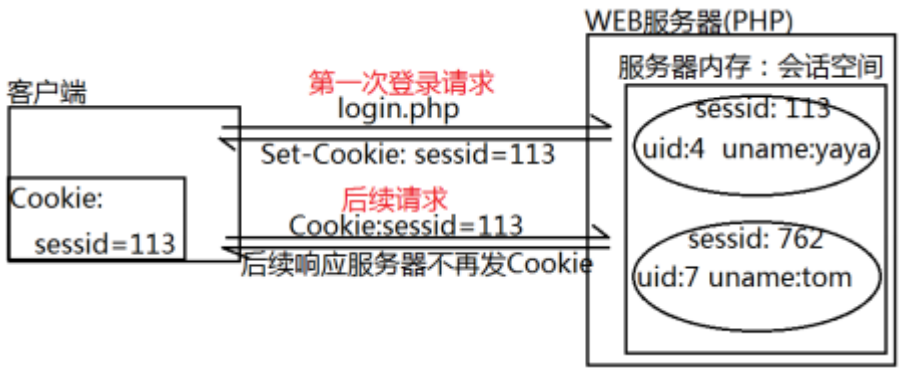
Access-Control-Allow-Methods: 服务器允许哪些请求方法

Access-Control-Allow-Headers: 服务器允许哪些请求头部

Access-Control-Allow-Credentials: true 服务器允许客户端提交身份认证信息

难点&面试题：跨域会话问题

服务器端会话的原理：



(1) 客户端浏览器第一次请求时，请求消息中无Cookie:sessid；

(2) 服务器在给出响应消息时，顺带在服务器端分配出Session空间并保存数据，

　　响应消息头部中，使用Set-Cookie: sessid=xxx将会话编号发送给客户端

(3) 浏览器将服务器端发来的会话编号保存在客户端

(4) 浏览器再次发送请求时，在请求头部中会携带自己的会话编号Cookie:sessid=xxx

(5) 服务器读取客户端提交的会话编号，识别出该编号之前存储的数据，输出响应消息，但不再发送新SessionID

结论：服务器端Session可用于识别出不同的用户（用户识别），

　　最关键字的点是客户端保存着Cookie:SESSID　—— 身份认证信息(Credentials)

提示：异步跨域请求时，浏览器默认都不会向服务器发送任何Cookie

——即身份认证信息不会发给服务器

——Session即失效

```
//  Angular中：
 this.http.get(url,  {
   withCredentials: true
 });
//  jQuery中：
$.ajax(
  url: url,
  withCredentials:  true,
  success: fn  )
//  原生 JS：
var  xhr = new XMLHttpRequest();
xhr.withCredentials = true
```

注意：客户端请求若“携带了身份认证信息”，则服务器端响应消息头“允许的跨域来源”就不允许用星号，

例如：Access-Control-Allow-Origin: *；

必须明确指定哪些客户端允许跨域访问，例如：Access-Control-Allow-Origin:http://127.0.0.1:4200；

同时还必须声明响应头：Access-Control-Allow-Credentials:true，以允许客户端请求携带身份认证信息。