这样写 Angular 应用：用 Angular 扩展语法编写 HTML 模板， 用组件类管理这些模板，用服务添加应用逻辑， 用模块打包发布组件与服务。

然后，我们通过引导根模块来启动该应用。 Angular 在浏览器中接管、展现应用的内容，并根据我们提供的操作指令响应用户的交互。



 Angular 应用中的 8 个主要构造块：

* [模块 (module)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "modules)：是一个带有@NgModule装饰器的类；
* [组件 (component)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "components)：组件负责控制屏幕上的一小块视图；
* [模板 (template)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "templates)
* [元数据 (metadata)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "metadata)：告诉 Angular 如何处理一个类；
* [数据绑定 (data binding)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "data-binding)
* [指令 (directive)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "directives)
* [服务 (service)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "services)：是实现专一目的的逻辑单元，提供了一些特定的功能，通常使用在组件内部作为组件功能的一个扩展。
* [依赖注入 (dependency injection)](https://angular.cn/docs/ts/latest/guide/architecture.html" \l "dependency-injection)：是组件引入外部构建（如服务）的一种机制。

## 组件

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

@Component({ //装饰器

selector: "hello", //元数据

//“hello” 可以匹配到html中的 <hello></hello>标签

template: "<p>{{greeting}}</p>"

//把模板放到选定的标签中，组件渲染的结果是：

//<hello>

// <p>Hello, Angular 2!</p>

//</hello>

})

export class HelloComponent { //组件类

private greeting: string; //变量的声明 变量名：类型

constructor() {

this.greeting = "Hello, Angular 2!";

}

}

装饰器是TypeScript所提供的一种语法特性，用来修饰一个类。



仅定义一个类，Angular2是不知道如何解释这一个类的，但如果在类上加上@Component装饰器，这个装饰器会在运行的时候把它的元数据参数通过某一种方式注入到类里面。这样Angular2就能识别出来这个组件类。所以装饰器的作用就是赋予一个类更丰富的信息（元数据）。

## 数据绑定

1. 插值语法：直接使用组件类里面的成员变量，如greeting
2. 属性绑定 [value] 把组件类里的属性传递到组件模板中

<input [value]="myData" />

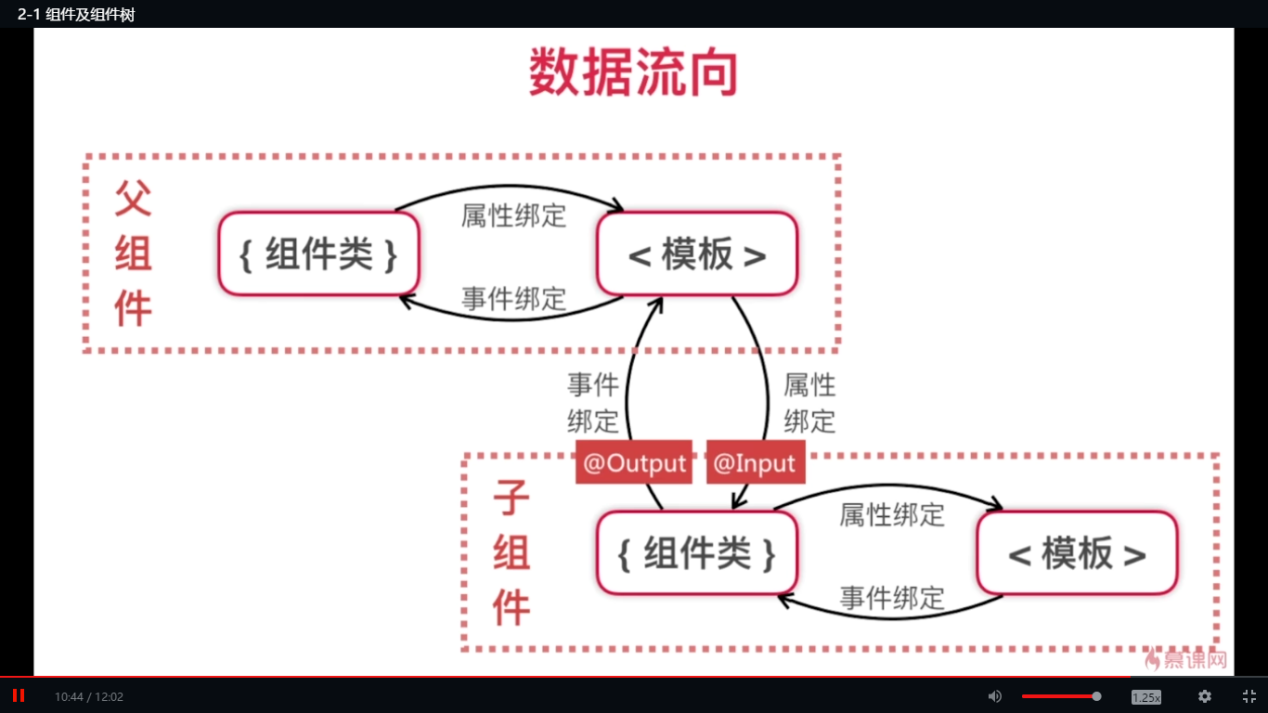
属性绑定既负责组件类内部属性和模板间的数据传递，同时也担负着组件之间的数据传递重任。

1. 事件绑定 (keyup) 把模板产生的数据通过函数调用的方式传递到组件类

<input (keyup)="handle($event)" />

1. 双向绑定 [(ngModel)]

<input [(ngModel)]="myData" />



## 指令



## 模块

模块有两层含义：

* 框架代码以模块形式组织（文件模块）
* 功能单元以模块形式组织（应用模块）

1. 文件模块其实就是TypeScript提供的语言层面的模块功能

* 核心模块 @angular / core
* 通用模块 @angular / common
* 表单模块 @angular / forms
* 网络模块 @angular / http

使用方法：只需要import即可

import {Http} from “@angular/http”

装饰器都是需要先导入才能够使用的

//@Component 装饰器

import {Component} from “@angular/core”

//@Directive装饰器

import {Directive} from “@angular/core”

import {ElementRef, Renderer} from “@angular/core”

1. 应用模块就是对应用类零散的组件、指令、服务等按功能进行归类包装。一个组件可以任意使用同模块的指令/组件，但是跨模块里的组件和指令则不能直接相互使用。如果要跨模块使用，就要通过“导入/导出”功能。

声明模块用的是**@NgModule**装饰器

@NgModule ({

declarations: [ //包装组件或指令

AppComponent,

SomeDirective

],

providers: [LoggerService], //依赖注入

imports: [OtherModule], //导入其他模块

bootstrap: [AppComponent], //指定根组件，该属性仅在根模块使用

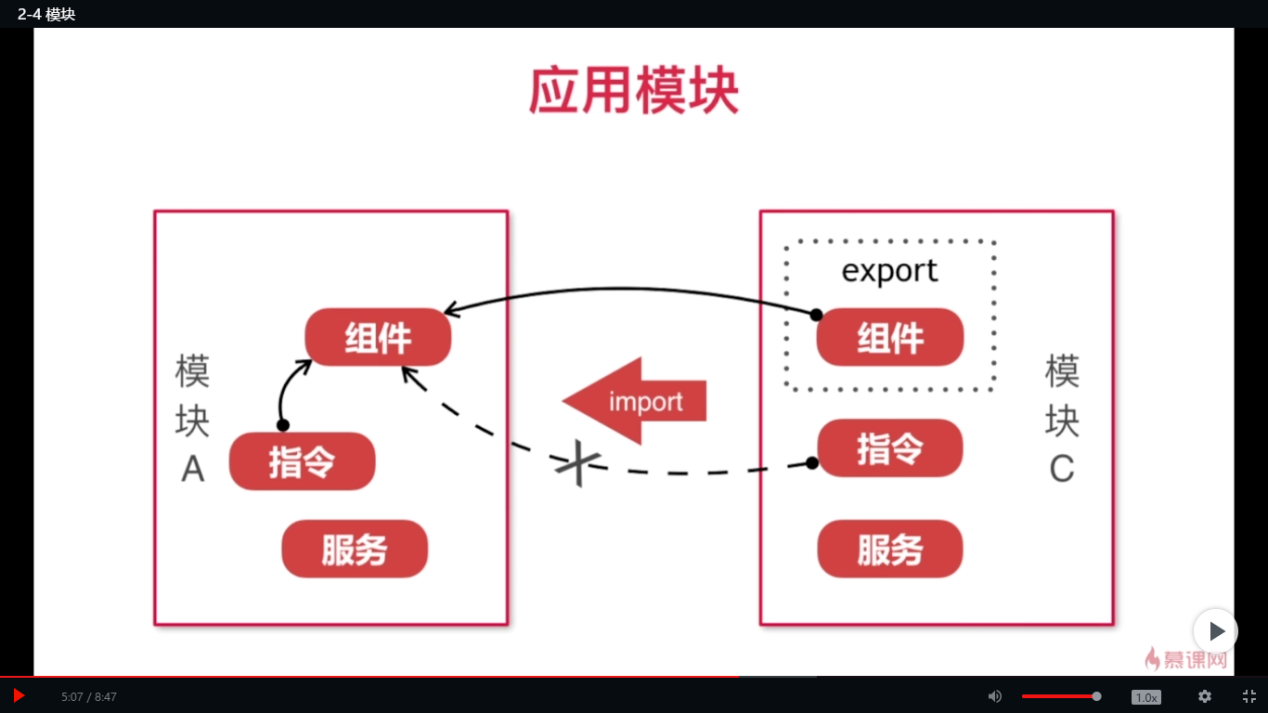
exports: [SomeDirective] //设定该模块对外暴露的组件或指令

})

export class AppModule {}

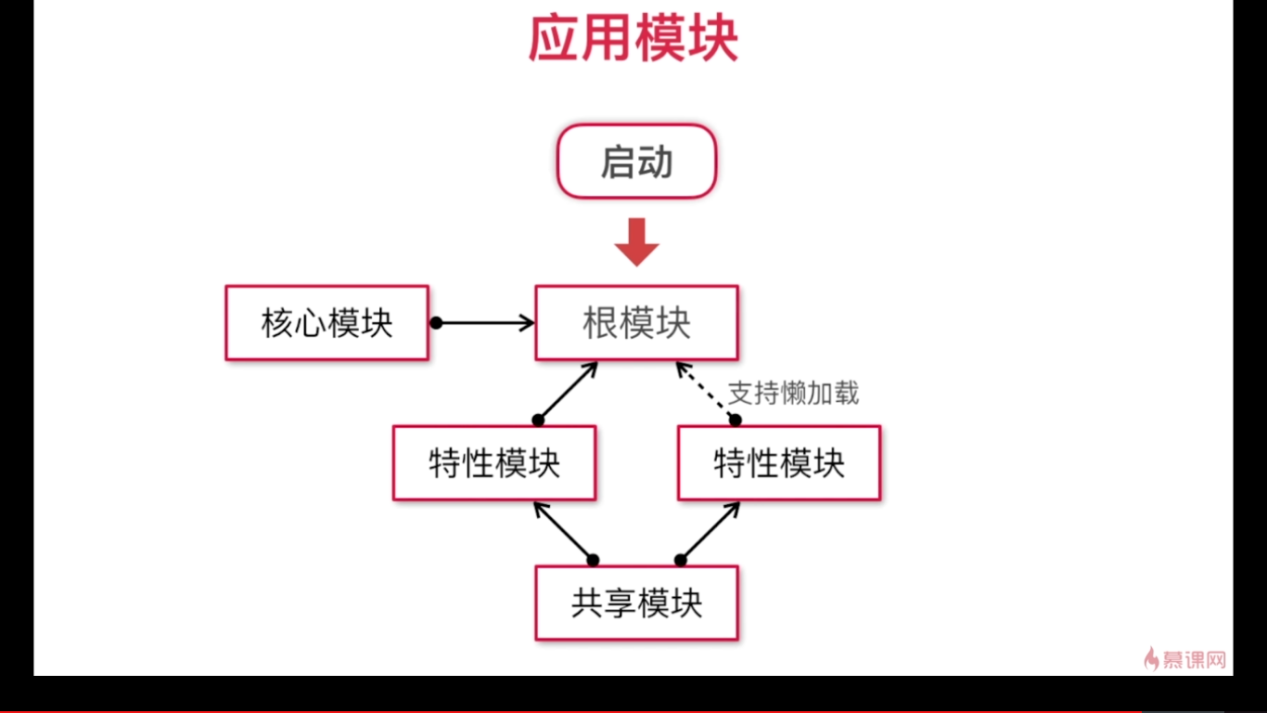
依赖注入能够作用在组件里，也能够作用在模块上，使用方法大致相同，区别在于作用域。作用到模块里的服务可以在**全局**使用，而注入到组件里的仅能在该组件及其子组件使用。

Angular2的模块既可以对外暴露一些组件/指令，同时又有一定的封闭性，能够隐藏内部的一些实现。



注意：服务并不在导入、导出的范畴内。

Angular2要想成功运行，至少需要定义一个模块，因为需要一个模块作为整个应用启动的入口，这样的模块就称为“根模块”。



接着，应用会不断添加新的功能，这些新的功能可以封装到一个新的模块里，然后导入到根模块里即可。这些新增的模块就称为“特性模块”。有了特性模块，根模块原来承载的功能逻辑也可以抽离出来放到某个特性模块里，这样就可以使根模块保持简洁。

“懒加载”特性非常有用，即使应用变得非常庞大，但初始加载的包体依然可以控制在一个合理的大小范围内，大大减少首屏加载的时间。

特性模块越来越多，把它们拥有的相似的功能/组件抽取出来，封装成一个独立的模块，这样的模块在逻辑意义上不能称为特性模块，可以称为“共享模块”。

还有一种模块类型称为“核心模块”。一个应用总有一些全局的组件或服务，它们只需要在应用启动的时候初始化一次即可，例如维护登陆信息的服务、公共的头部/底部组件等，把这些放到“核心模块”。它只导入到根模块，尽量不要导入到其他模块，避免出现一些不可预料的结果。

应用的各个特性被切分成大大小小的模块，逻辑结构非常清晰，处于总指挥地位的根模块非常简洁，没有繁琐的业务细节。

使用双向数据绑定需要引入三个模块：

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

import { BrowserModule } from '@angular/platform-browser';

import { FormsModule } from '@angular/forms'; /

## 服务

实现专一目的的逻辑单元，通常使用在组件内部，作为组件功能的扩展，那么服务怎样才能被组件所使用呢？

## 依赖注入

组件引入外部构建（如服务）的一种机制。组件引入服务实际上引用的是该服务类的实例，所以服务在被引入之前会有一个实例化的过程，并且这个实例要被缓存起来，以备其他组件使用。这些都是靠依赖注入机制实现的。服务的实例存储在依赖注入机制建立的注入器对象里，当组件需要依赖某个服务的时候，依赖注入机制会从注入器对象里查找匹配的实例，找到后便执行注入操作。

依赖注入的关键步骤是在@Component里加入providers属性。

@Component({

selector: “hello”,

template: “<p>{{greeting}}</p>”,

**providers**: [LoggerService]

})

export class HelloComponent {

private greeting: string;

constructor(logger: LoggerService) {

this.greeting = “Hello, Angular2!”;

logger.debug(“finished”);

}

}

首先依赖注入机制会根据**providers**提供的LoggerService类预先实例化一个对象，并缓存到注入器里。在contructor里也有一个LoggerService类型的参数logger。接下来，依赖注入机制会根据contructor里的参数需求从注入器对象里尝试查找LoggerService的实例，找到后传入到contructor里。最终组件内部获得了LoggerService的实例引用。

BrowserModule模块包含了浏览器上启动应用的一些关键逻辑，所以必须在应用的根模块里面引入。