### SpringBoot特点

* 遵循“约定优于配置”的原则，只需要很少的配置或使用默认的配置。能够使用内嵌的Tomcat、Jetty服务器，不需要部署war文件。
* 提供定制化的启动器Starters，简化Mavern配置，开箱即用。
* 纯Java配置，没有代码生成，也不需要XML配置。
* 提供了生产级的服务监控方案，如安全监控、应用监控、健康检测等。

config、controller、mapper、pojo、service 这几个模块常见于 Spring Framework 和其他 Java Web 开发框架中。它们各自有不同的职责，相互协作完成一个完整的 Web 应用程序。

1. config

通常用于存放配置文件和配置类。这里保存了应用程序的全局配置，例如数据库连接、安全性配置、应用程序的初始化参数等。在 Spring Boot 中，我们常常使用 @Configuration 注解来声明一个配置类，并使用 @Bean 注解来定义和初始化一个对象，该对象会被添加到 Spring 的 IoC 容器中进行管理。

2. controller

控制器层，负责处理用户的请求并返回响应。它接收到 HTTP 请求后，通常会调用 service 层处理业务逻辑，然后返回视图（HTML、JSON 等）。在 Spring MVC 中，我们使用 @Controller 或 @RestController 注解来定义一个控制器。

3. mapper

数据映射层，通常与数据库打交道。在 MyBatis 或 Hibernate 这样的 ORM 框架中，mapper 通常是一个接口，用于定义 SQL 查询如何映射到 Java 对象以及 Java 对象如何映射到数据库表。它的主要作用是进行数据库操作和数据的持久化。

4. pojo

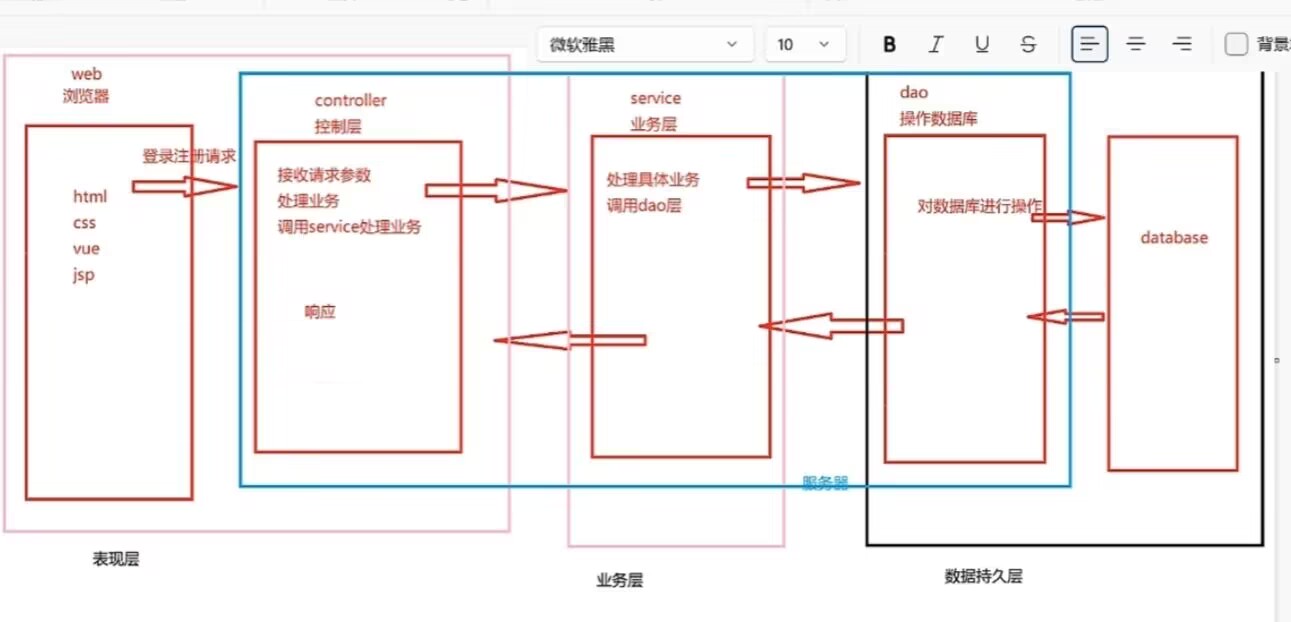
POJO (Plain Old Java Object) 是一个简单的 Java 对象，通常用于封装数据。它通常只包含私有字段、公有的 getter 和 setter 方法、构造函数、hashcode、equals、toString 方法等。POJO 通常用作数据传输对象（DTO）来在应用的各层之间传递数据。

5. service

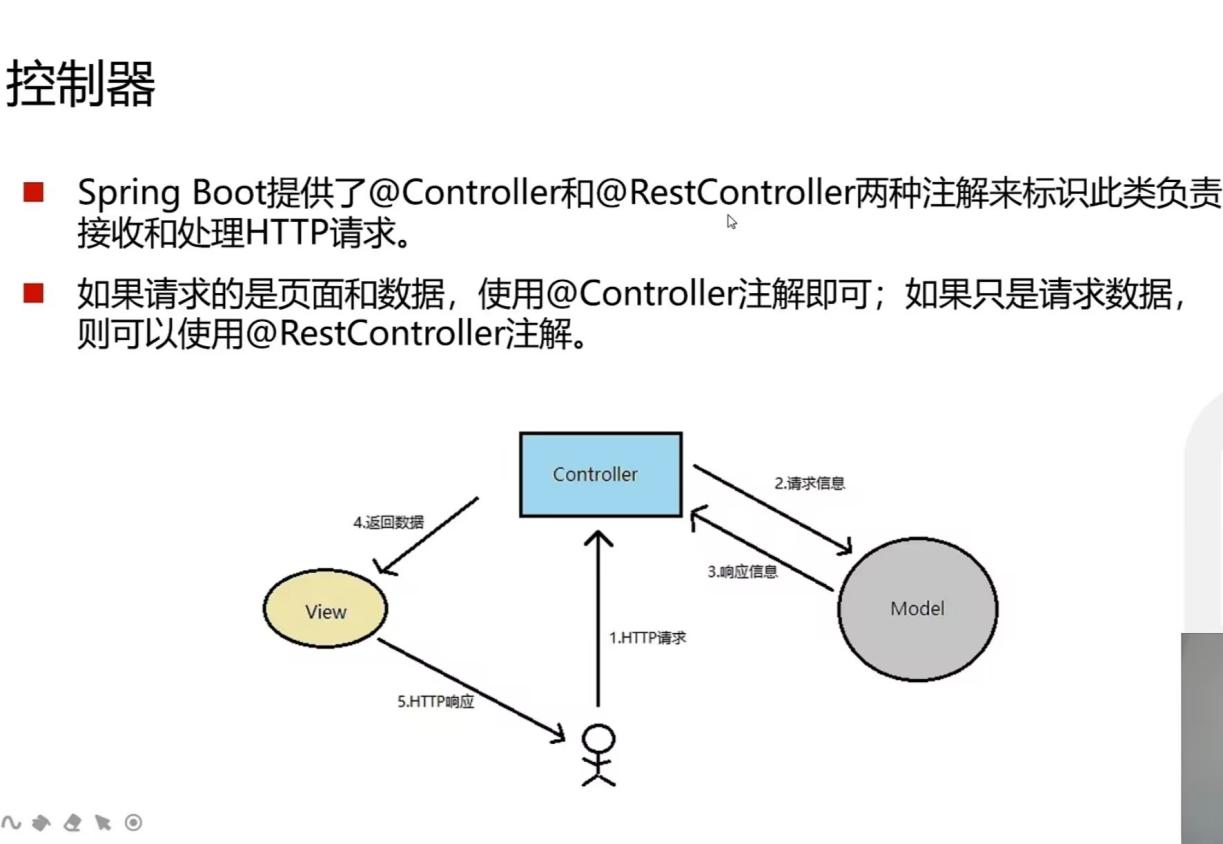
服务层，用于实现业务逻辑。它通常会调用 mapper 层来进行数据库操作，并处理更复杂的业务逻辑。服务层可以被控制器层调用，并返回处理结果。在 Spring 中，我们使用 @Service 注解来定义一个服务。

联系：

通常，用户的请求首先到达 controller 层。controller 层会调用 service 层来处理业务逻辑。service 层则会与 mapper 层交互，进行数据库的 CRUD 操作。数据通常以 POJO 的形式在各层之间传递。config 层则提供全局的配置和支持。这些层次结构共同构成了一个完整的 Web 应用程序。



**控制器**

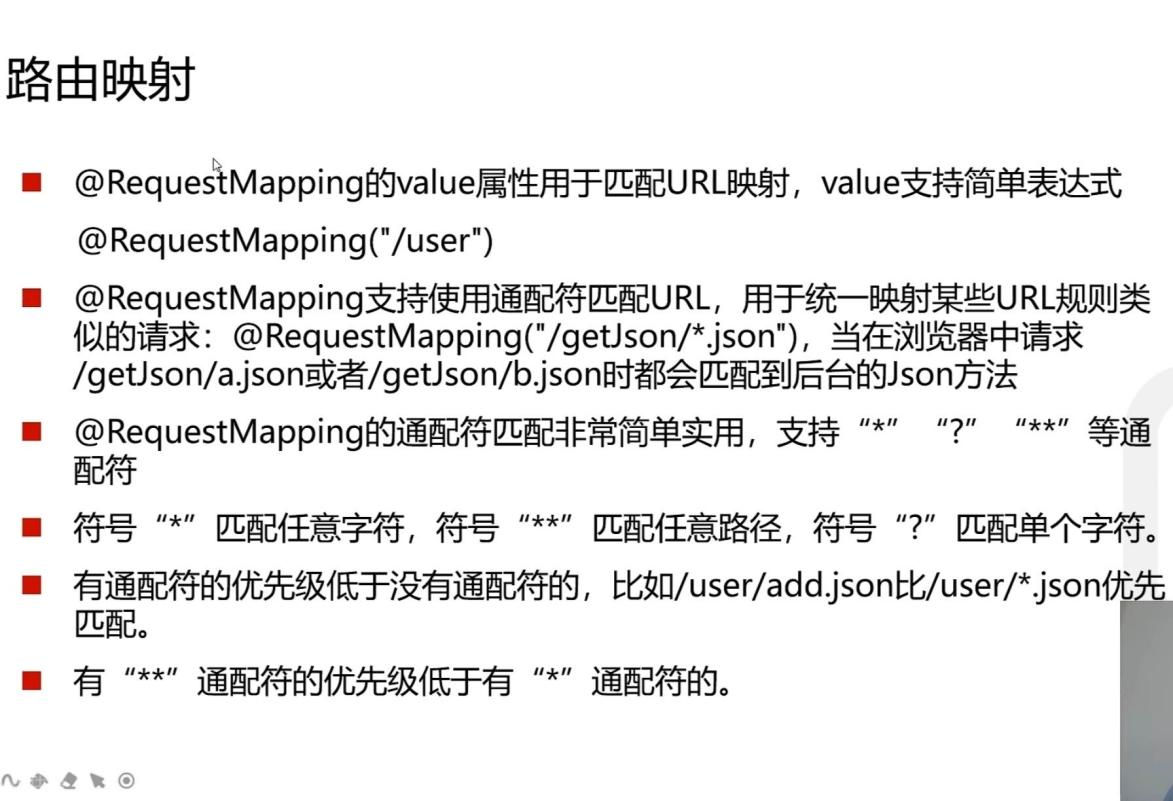


**@RestController的用法**



**路由映射**

* RequestMapping注解主要负责URL的路由映射。它可以添加在Controller
* 类或者具体的方法上。
* 如果添加在Controller类上，则这个Controller中的所有路由映射都将会加上此映射规则，如果添加在方法上，则只对当前方法生效。
* @RequestMapping注解包含很多属性参数来定义HTTP的请求映射规则。常用的属性参数如下:
* value:请求URL的路径，支持URL模板、正则表达式
* method: HTTP请求方法



**Method匹配**



**使用Swagger生成Web API 文档**

**依赖**：

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.9.2</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.9.2</version>

</dependency>

**配置**Swagger （放在config里面）

@Configuration//告诉Spring容器，这个类是一个配置类

@EnableSwagger2 //启用Swagger2功能

public class Swagger2Config {

@Bean

public Docket createRestApi() {

return new Docket(DocumentationType.SWAGGER\_2)

.apiInfo(apiInfo())

.select()

// com包下所有API都交给Swagger2管理

.apis(RequestHandlerSelectors.basePackage( "com" ))

.paths(PathSelectors.any() ).build();

}

// AP工文档页面显示信息

private ApiInfo apiInfo() {

return new ApiInfoBuilder()

.title("演示项目API")//标题

.description("学习Swagger2的演示项目")//描述

//.version("1.0");

.build();

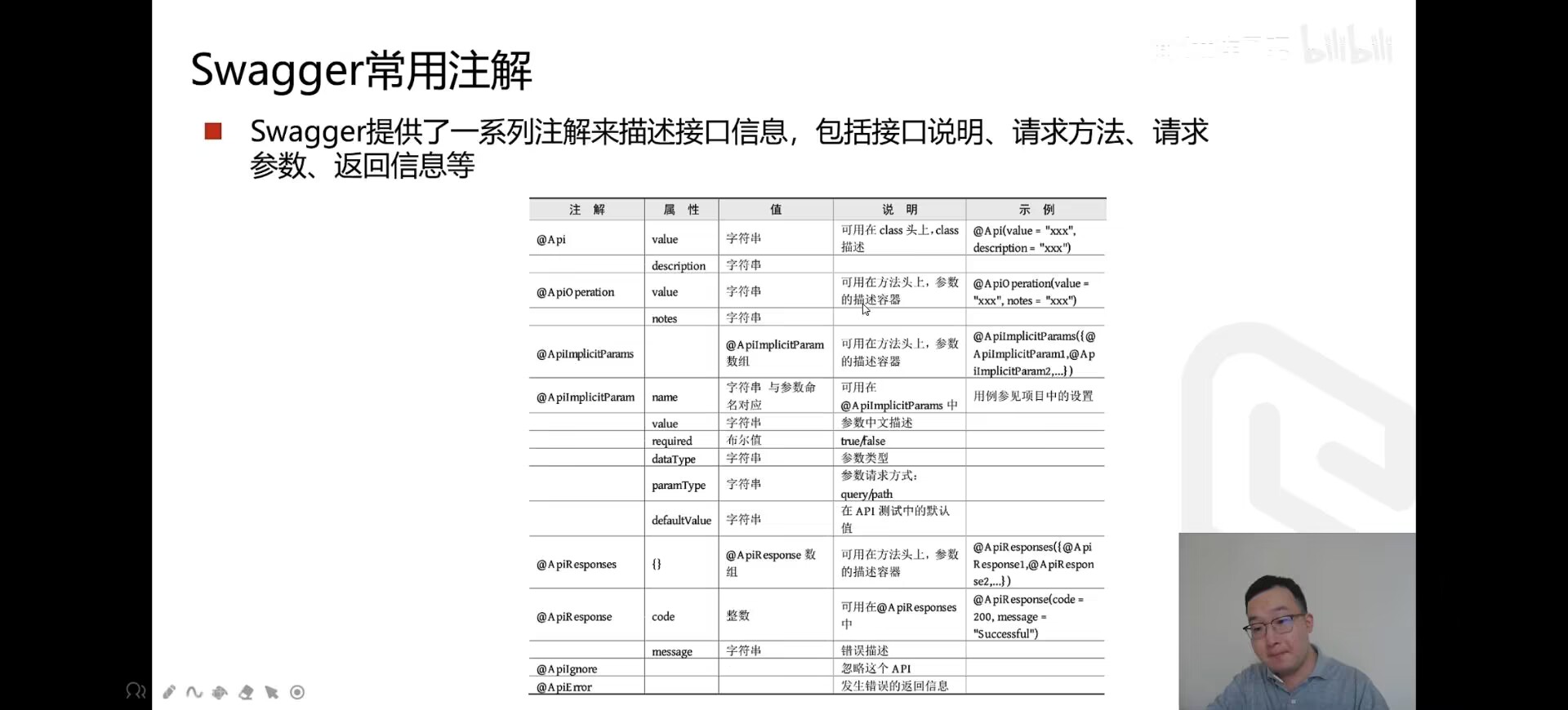
}

}

最后通过 http://127.0.0.1:8080/swagger-ui.html 可以查看接口api

**Swagger常用注解**

Swagger提供了一系列注解来描述接口信息，包括接口说明、请求方法、请求参数、返回信息等



**SpringBoot跨域问题**

SpringBoot跨域问题的根源在于浏览器的**同源策略**（Same-Origin Policy）。同源策略是浏览器的一种安全机制，**它限制了一个网页上的文档或脚本如何与不同源（即不同协议、主机或端口）的资源进行交互**。这种限制的目的是为了防止恶意网站执行可能危害用户的跨站脚本攻击（Cross-Site Scripting，XSS）。

在SpringBoot应用中，当我们尝试从一个源（例如域名A）的页面通过Ajax请求访问另一个源（例如域名B）的资源时，浏览器会因为同源策略的限制而阻止这种跨域请求，从而导致跨域问题的出现。

为了解决这个问题，SpringBoot提供了CORS（跨域资源共享，Cross-Origin Resource Sharing）支持。CORS是一种W3C标准，它允许服务器声明哪些源是合法的，从而使得跨域请求得以正常进行。在SpringBoot应用中，我们可以通过配置CORS来允许特定的源进行跨域请求，从而解决跨域问题。

**配置(可放在config里)**

//解决跨域问题

import org.springframework.context.annotation.Configuration;

import javax.servlet.\*;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@Configuration

public class CorsConfig implements Filter {

@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) res;

HttpServletRequest request = (HttpServletRequest) req;

String origin = request.getHeader("Origin");

if(origin!=null) {

response.setHeader("Access-Control-Allow-Origin", origin);

}

String headers = request.getHeader("Access-Control-Request-Headers");

if(headers!=null) {

response.setHeader("Access-Control-Allow-Headers", headers);

response.setHeader("Access-Control-Expose-Headers", headers);

}

response.setHeader("Access-Control-Allow-Methods", "\*");

response.setHeader("Access-Control-Max-Age", "3600");

response.setHeader("Access-Control-Allow-Credentials", "true");

chain.doFilter(request, response);

}

@Override

public void init(FilterConfig filterConfig) {

}

@Override

public void destroy() {

}

}



**HTTP状态码**

RESTfulAPI中使用HTTP状态码来表示请求执行结果的状态，适用于REST API设计的代码以及对应的HTTP方法。



**ORM介绍**

