拦截器 HandlerInterceptor

拦截器可以对控制器方法进行拦截操作

拦截器属于 SpringMVC 内置的功能， 不属于Spring框架范畴

我们在 配置 拦截器的时候 可能需要麻烦一点， 我们需要创建 MVC配置类

并向 MVC 添加拦截器

拦截器对 JSP 和 静态文件是没有效果的， 拦截器基于SpringMVC ， 所以只有在访问呢了后端控制器的时候

才会触发拦截器

创建拦截器 HandlerInterceptor

配置拦截器 WebMvcConfigurer

1. preHandle
    a. 用在 控制器方法执行之前执行
    b. 如果存在多个拦截器正序执行
2. postHandle
    a. 用在 控制器方法执行之后， 视图渲染之前执行
    b. 如果存在多个拦截器 倒序执行
3. afterCompletion
    a. 在前端控制器渲染视图之后执行
    b. 如果存在多个拦截器 倒序执行
4. 拦截器与过滤器区别
    a. 归属不同， 一个SpringMVC层的功能， 一个是Tomcat服务的功能
    b. 执行时机不同，
    c. 过滤器 执行时间 比 拦截器早的多！！！

HttpPutFormContentFilter 和 HiddenHttpMethodFilter

@ResponseBody 允许我们直接向 HTML页面输出内容

异常处理

异常， 当我们有一大坨代码， 里面可能有我们无法预知的问题的时候，

我们不可能 每一个细节 都去繁琐的 判断处理

这个时候 使用异常就是最好的 办法，

异常处理 比较消耗 系统性能

业务逻辑正常运行是不可能发生异常的

频繁使用异常， 没有问题的

- Spring MVC 异常处理
    a. 使用 Spring MVC 提供的简单异常处理器 SimpleMappingExceptionResolver。
    b. 实现 Spring 的异常处理接口 HandlerExceptionResolver，自定义自己的异常处理器。

c. 使用 @ExceptionHandler 注解实现异常处理

- SimpleMappingExceptionResolver 实现类的简单使用
  - 指定异常 <prop key="java.lang.ArithmeticException">/aria.jsp</prop>
  - 默认异常 <property name="defaultErrorView" value="/error.jsp" ></property>
- HandlerExceptionResolver 实现接口
  - 处理自定义异常
  - 返回MV 跳转指定页面
- 注解异常 使用注解处理类内的异常信息
  - @ExceptionHandler({ArithmeticException.class})

数据转换器

所谓的消息类型转换， 其实就是数据映射

其实就是把 外部的数据， 转换成 Java 可以识别的 Java 类型

- 消息转换器
  - 根据不同的Content-Type等情况,Spring-MVC会采取不同的 HttpMessageConverter实现来进行信息转换解析。
  - 前端以Content-Type 为application/json,传递json字符串数据; JsonbHttpMessageConverter
  - 前端以Content-Type 为multipart/form-data，传递数据； FormHttpMessageConverter
- 自定义消息转换器
  - 主要是处理 Converter 接口方法
  - 实例化对象并注入 或者 配置 Bean ConversionServiceFactoryBean
  - addFormatters(FormatterRegistry registry) 能够把我们的 消息转换器 添加到 SpringMVC上下文中!
  - <mvc:annotation-driven conversion-service="conversionServiceFactory> XML 方式

处理JSON消息

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.13.3</version>
</dependency>
```

Swagger-UI

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
```

```
 <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.36</version>
</dependency>
```

注意配置 addResourceHandlers 放行

http://localhost:8080/应用/v2/api-docs

registry.addResourceHandler("/swagger/**").addResourceLocations("/swagger/");

Swagger 注解

Swagger 会去扫描SwaggerConfig 中配置的包路径下的带有Swagger 注解的类文件，并最后生成一串扫描的Json文件

我们访问 v2/api-docs 可以查看信息

@Api :用在类上，说明该类的作用,需要说明的是较老的版本用的value表示扫描生成的类名，1.5后要用tag 表示类名

- @Api(tag= "UserController", description = "用户相关api")

@ApiOperation :用在方法上，说明方法的作用

- @ApiOperation(value = "查找用户", notes = "查找用户", httpMethod = "GET", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)

@ApiParam： 用在参数列表中，表明参数的含义

- @ApiParam(value = "创建或更新距离当前时间(月)") Integer time

@ApiImplicitParams :用在方法上包含一组参数说明

- **@ApiImplicitParam :**用在@ApiImplicitParams注解中，指定一个请求参数的各个方面

```
@Configuration
@EnableSwagger2
@ComponentScan("com.lovecoding.mvc")
public class SwaggerConfig {

    @Bean
    public Docket getDocket() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.any())
                .paths(PathSelectors.any())
```

```
                .build();
    }
    private ApiInfo apiInfo() {
        ApiInfo apiInfo = new ApiInfo(
                "对接服务平台API文档", //标题
                "", //描述
                "1.0", //版本
                "",
                "",
                "", //签名
                "" //签名链接
        );
        return apiInfo;
    }
}
```