SpringMV

- SpringMVC是web层框架
- 五大组件 -- 面试考点

```
DispatcherServlet -- 前端控制器
HandlerMapping  -- 处理映射器
Controller
ModelAndView  -- 视图数据模型
ViewResolver  -- 视图解析器
```

- SpringMVC处理请求的过程

```
1. 请求到达DispatcherServlet
2. 调用HandlerMapping进行解析
3. 到达对应的controller中的对应方法，controller处理请求
4. 由Controller返回ModelAndView对象给DispatcherServlet，其中封装了视图和数据
5. DispatcherServlet调用ViewResovler，解析视图，之后，进行视图渲染
```

- SpringMVC中的常用注解

```
1. @RestController :@Controller  @ResponseBody
2. @ResponseBody: 作用：使用该注解则不再调用ModelAndView以及视图解析器，而会自动调用
SPring的组件MessageConvert,将返回的数据转换为某种特定格式（json，xml），装入响应体
3. @RequstMapping
   @GetMapping  -- 只接受get请求
   @PostMapping -- 只接受post请求

4. @RequestBody -- 从请求中接受json数据
    -- 也会调用信息转换器，将json数据转换为对象数据
```

- HttpMessageConvert

```
SpringBoot自动配置好的
配置的信息转换器有：
    stringHttpMessageConverter
    mappingJackson2HttpMessageConverter

/**
  * 测试SpringBoot自动配置好的HttpMessageConvert有哪些
  */
@Test
void contextLoads() {
    String[] names =
context.getBeanNamesForType(HttpMessageConverter.class);
    for (String name:names){
        System.out.println(name);
    }
}
```

有环境和准备类的项目案例：

https://gitee.com/huadahua/springmvc-rest.git

有上课代码的项目案例：

https://gitee.com/huadahua/spring-mvc-rest-hascontent.git

- SpringBoot自动配置的内容

```
设置一个DispatcherServlet
设置内部配置以支持控制器
设置默认的资源位置（images，CSS，JavaScript）
设置默认的Message Converters
还有更多，更多
```

常用传参的三种方式

1. 键值对参数

```
get请求： http://....login?name=xx&pwd=xx
post请求： 表单传参，也是键值对传参  xx=xx&xx=xx

控制器中接受键值对参数需要使用注解@RequestParam
   @PostMapping("/login")
    public String login(@RequstParam("username") String name,
@RequestParam("password") String pwd){
        System.out.println(name+"--"+pwd);
        return "登录成功！";
    }
注意：若请求中的key与参数列表的参数名称一致，则该注解中的value可以省略，甚至该注解可以省略

http client工具中发起get和post请求并测试
 ###
GET http://localhost:8080/users/login?name=jack&pwd=1234

<> 2022-06-26T152817.200.txt
<> 2022-06-26T152558.200.txt

###
POST http://localhost:8080/users/login
Content-Type: application/x-www-form-urlencoded

name=rose&pwd=hello

若http client找不到（tools不存在），可以直接创建要给以.http结尾的文件进行http请求测试
```

2. json传参

```
前端传递给后端的是json数据{key:val,....}

控制器如何接受json数据并将其转换为对象数据
    通过注解@RequestBody，作用与参数前，表示接受json数据并自动调用信息转换器转换为对象

json传参不适用于get请求，因为get请求参数位于路径中

/**
     * 演示json传参
     */
```

```
    @PostMapping("/regist")
    public String regist(@RequestBody User user){
        System.out.println(user);
        return "注册成功!";
    }

http client中：
POST http://localhost:8080/users/regist
Content-Type: application/json

{
  "id": 1,
  "name": "张三",
  "age": 23
}
```

3. 路径传参

```
Get  /users/login/{name}

控制器从路径中提取参数，通过使用注解@PathVariable来提取，该注解作用于参数列表的参数前

@GetMapping("/deleteUserById/{pid}")
public String delteUserById(@PathVariable("pid") Integer id){
    System.out.println(id);
    return "删除成功！";
}

若占位符{}中的名称与参数名称相同，则注解中的值可以省略
```

RestFul

```
RESTFUL是一种应用程序的设计风格和开发方式，基于HTTP协议 -- 支持很多请求方式 - get，post，put，delete。。。。。

核心思想，通过请求方法来表明进行的操作，
eg: 新增 -- 发起POST请求
    删除 -- 发起DELETE请求
    修改 -- 发起PUT请求
    查询 -- 发起GET请求



    eg:以前的做法：
        UserController中
            @RequestMapping("/user/addUser")
            void addUser(..){..}

            @RequestMapping("/user/deleteUser")
            void deleteUser(..){..}

            @RequestMapping("/user/findUserById")
            void findUserById(..){..}

            @RequestMapping("/user/updateUser")
            void updateUser(..){..}
```

```
使用restFul：
    @PostMapping("/user")
    void addUser(..){..}

    @DeleteMapping("/user")
    void deleteUser(..){..}

    @GetMapping("/user")
    void findUserById(..){..}

    @PutMapping("/user")
    void updateUser(..){..}

但是，这种使用并不多，表单中只支持GET,POST请求，不支持其他请求，目前restFul风格严格遵守的很少，
eg：修改操作，其实应该先查询，再修改，这里即涉及查询，又涉及修改，所以不适合使用restful风格，
Restful风格适用的场景： 如果是单纯操作数据，可以使用restful，ES中严格遵守restful
```

- @ResponseStatus -- restful风格中使用 -- 了解

```
作用：修改响应对象中的响应状态码

http协议中：
    response有状态码，经常见的有200
根据restful约定，不用的请求应返回不同的响应码；
    restful的规范：
        GET 请求，响应码应为200
        PUT 请求，响应码应为204(NO_CONTENT)
        POST请求，响应码应为201(CREATED)
        DELETE请求，响应码应为204(返回空响应)

该注解作用的位置：
   可以作用于controller方法上方，修改响应的状态码
   也可用于异常处理类的方法上方，抛出不同异常，返回不同的响应码
使用HttpStatus中的枚举类型值，
    eg:HttpStatus.NO_CONTENT -- 204
       HttpStatus.CREATED -- 201
```

- ResponseEntity对象的使用

执行操作--下载文件--异步下载，此时响应体中的数据既不是json，也不是字符串数据，而是文件中数据，响应对象中的header部分的Content-typeye 也必须做出修改，响应对象中的状态码也可能需要修改，此时出现了需要对响应对象大幅度修改，响应体修改，响应头也要修改，如何做？

@ResponseStatus只能修改响应对象的状态码

作用：

使用ResponseEntity对象，可以处理响应体，响应头，响应状态码

用法：

使用ResponseEntity作为controller的返回值，我们可以方便地处理响应的header，状态码以及body。而通常使用的@ResponseBody注解，只能处理body部分。

使用场景：

这也是为什么通常在下载场景中会使用ResponseEntity，因为下载需要设置header里的content-type以及特殊的status

ResponseEntity的优先级高于@ResponseBody。在不是ResponseEntity的情况下才去检查有没有@ResponseBody注解。如果响应类型是ResponseEntity可以不写@ResponseBody注解，写了也没有关系。

RestTemplate对象

- 用于在向其他服务发起请求的，用法和TestRestTemplate一样

  用于服务之间发起和接受请求，提供了支持restful的各种方法
  支持使用RequestEntity和ResponseEntity，支持发送自定义请求，接受自定义响应

  - 模拟考试 -- 开放一周，去做，下周上课留时间回答不会的题目
- @RequestHeader

  作用于方法的参数前的，表示从请求的header部分提取参数
  经常在服务器端获取请求header中的User-Agent的值：
  该属性的值包括：客户端的信息，eg:操作系统，浏览器。。。。。