### JAVA第三阶段—DAY08-JAVA案例

1. 拦截器案例

1）先不用登录直接访问后台页面，引出拦截器的作用

2）登录时，将用户对象保存到session中

3）编写拦截器，实现拦截逻辑

4）配置拦截器

实现HandlerInterceptor接口

/\*\*  
\* 登录拦截器  
\*/  
public class LoginInterceptor implements HandlerInterceptor {  
​  
  /\*\*  
    \* 前置处理  
    \*/  
  @Override  
  public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
​  
      //查询session是否保存用户  
      User user = (User) request.getSession().getAttribute("user");  
      if(user == null){  
          //没有登录，重定向到login页面  
          response.sendRedirect("/pages/login");  
          //返回false，拦截  
          return false;  
      }  
      //返回true，放行  
      return true;  
  }  
}  
​

配置拦截器

<!--配置拦截器-->  
  <mvc:interceptors>  
      <mvc:interceptor>  
          <!--配置拦截的URL请求路径 /\*\*代表所有请求-->  
          <mvc:mapping path="/\*\*"/>  
          <!--配置不拦截的请求路径-->  
          <mvc:exclude-mapping path="/\*\*/login"/>  
          <mvc:exclude-mapping path="/\*\*/\*.css"/>  
          <mvc:exclude-mapping path="/\*\*/\*.js"/>  
          <mvc:exclude-mapping path="/\*\*/\*.png"/>  
          <mvc:exclude-mapping path="/\*\*/\*.jpg"/>  
          <!--配置class-->  
          <bean class="com.blb.bookms.interceptor.LoginInterceptor"/>  
      </mvc:interceptor>  
  </mvc:interceptors>

1. 上传案例
2. 引入相关依赖
3. 配置上传处理器
4. 编写上传控制器
5. 编写上传页面

导入依赖

      <!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->  
      <dependency>  
          <groupId>commons-fileupload</groupId>  
          <artifactId>commons-fileupload</artifactId>  
          <version>1.4</version>  
      </dependency>  
​  
      <!-- https://mvnrepository.com/artifact/commons-io/commons-io -->  
      <dependency>  
          <groupId>commons-io</groupId>  
          <artifactId>commons-io</artifactId>  
          <version>2.4</version>  
      </dependency>

2）配置上传处理器

  <!--配置文件上传处理器-->  
  <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
      <!--默认编码-->  
      <property name="defaultEncoding" value="UTF-8"></property>  
      <!--限制文件上传的大小 5m-->  
      <property name="maxUploadSize" value="5242880"></property>  
  </bean>

3）上传方法

方法参数使用CommonsMultipartFile来定义上传文件

@RequestMapping("/saveBook")  
public String saveBook(Book book, @RequestParam(value = "pic",required = false)CommonsMultipartFile file){  
  //保存文件的路径  
  String dir = "C:\\tomcat\\apache-tomcat-8.0.33\\webapps\\images\\";  
  if(file != null){  
      //获得原始文件名  
      String filename = file.getOriginalFilename();  
      String suffix = filename.substring(filename.lastIndexOf("."));  
      //创建新文件名  
      filename = UUID.randomUUID().toString().replace("-","") + suffix;  
      //保存文件名  
      book.setBookImage(filename);  
      //创建文件对象  
      File imgFile = new File(dir+filename);  
      //保存上传文件到服务器磁盘上  
      try {  
          file.transferTo(imgFile);  
      } catch (IOException e) {  
          e.printStackTrace();  
      }  
  }  
  System.out.println("book:"+book);  
  if(book.getState() == null){  
      book.setState(0);  
  }  
  bookService.saveBook(book);  
  return "redirect:/pages/index";  
}

4） 表单

form标签要添加enctype="multipart/form-data"，文件域的name不能和实体类的属性一样，否则会出现错误，这里使用了pic

<form action="/book/saveBook" method="post" enctype="multipart/form-data">  
  ...  
  <input type="file" name="pic" >  
  ...  
</form>

1. 下载案例

编写下载文件的方法，演示浏览器下载

/\*\*  
\* 下载控制器  
\*/  
@Controller  
public class DownloadController {  
​  
  @RequestMapping("download")  
  public void download(String file, HttpServletResponse response) throws IOException {  
      //下载文件的路径  
      String path = "D:\\install\\";  
      File downFile = new File(path+file);  
      if(downFile.exists()){  
          //设置浏览器下载内容类型，响应头  
          response.setContentType("application/x-msdownload");  
          response.setHeader("Content-Disposition","attachment;filename="+file);  
          //通过流发送文件  
          Files.copy(downFile.toPath(),response.getOutputStream());  
      }  
  }  
}

1. Restful案例

编写对于单表的Restful增删改查接口

用postman测试

@RestController

public class BookController {

@Autowired

private BookService bookService;

@GetMapping("/books")

public JsonResult findAll(){

try {

List<Book> list = bookService.list();

return new JsonResult(1,list);

}catch (Exception ex){

ex.printStackTrace();

return new JsonResult(0,null);

}

}

@GetMapping("/book/{id}")

public JsonResult findById(@PathVariable("id")Integer id){

try {

Book book = bookService.getById(id);

return new JsonResult(1,book);

}catch (Exception ex){

ex.printStackTrace();

return new JsonResult(0,null);

}

}

@PostMapping("/book")

public JsonResult saveBook(@RequestBody Book book){

try {

bookService.save(book);

return new JsonResult(1,null);

}catch (Exception ex){

ex.printStackTrace();

return new JsonResult(0,null);

}

}

@PutMapping("/book")

public JsonResult updateBook(@RequestBody Book book){

try {

bookService.updateById(book);

return new JsonResult(1,null);

}catch (Exception ex){

ex.printStackTrace();

return new JsonResult(0,null);

}

}

@DeleteMapping("/book/{id}")

public JsonResult delBook(@PathVariable("id")Integer id){

try {

bookService.removeById(id);

return new JsonResult(1,null);

}catch (Exception ex){

ex.printStackTrace();

return new JsonResult(0,null);

}

}

}