35_阶段测试：验证基于 Spring AOP 机制能否优化异常跳转错误页面问题

1、开篇

上节课介绍了利用 Spring AOP 实现全局异常处理的思路，从 ControllerAdvice 与 ExceptionHandler 注解入手，我们创建了自己的全局异常处理类 GlobalExceptionHandler。到 Controller 中抛出异常的时候 GlobalExceptionHandler 中有对应的方法就可以捕获到。针对不同的异常类，可以编写不同的异常处理方法。这节课我们一起来验证 GlobalExceptionHandler 类能否处理 Controller 中抛出的异常信息，从而解决页面报错的问题。今天的内容：

- 打包服务
- 上传服务
- 启动服务

2、打包服务

由于上节课已经将全局异常处理代码和抛出异常的代码完成了，这里直接对代码进行打包。如图 2 所示，打开我们熟悉的酒店管理后台应用服务，在 IntelliJ IDEA 中选择下方的“Teminal”按钮，在显示的命令行中（红色箭头的地方）输入对应的命令，用来对项目进行打包。

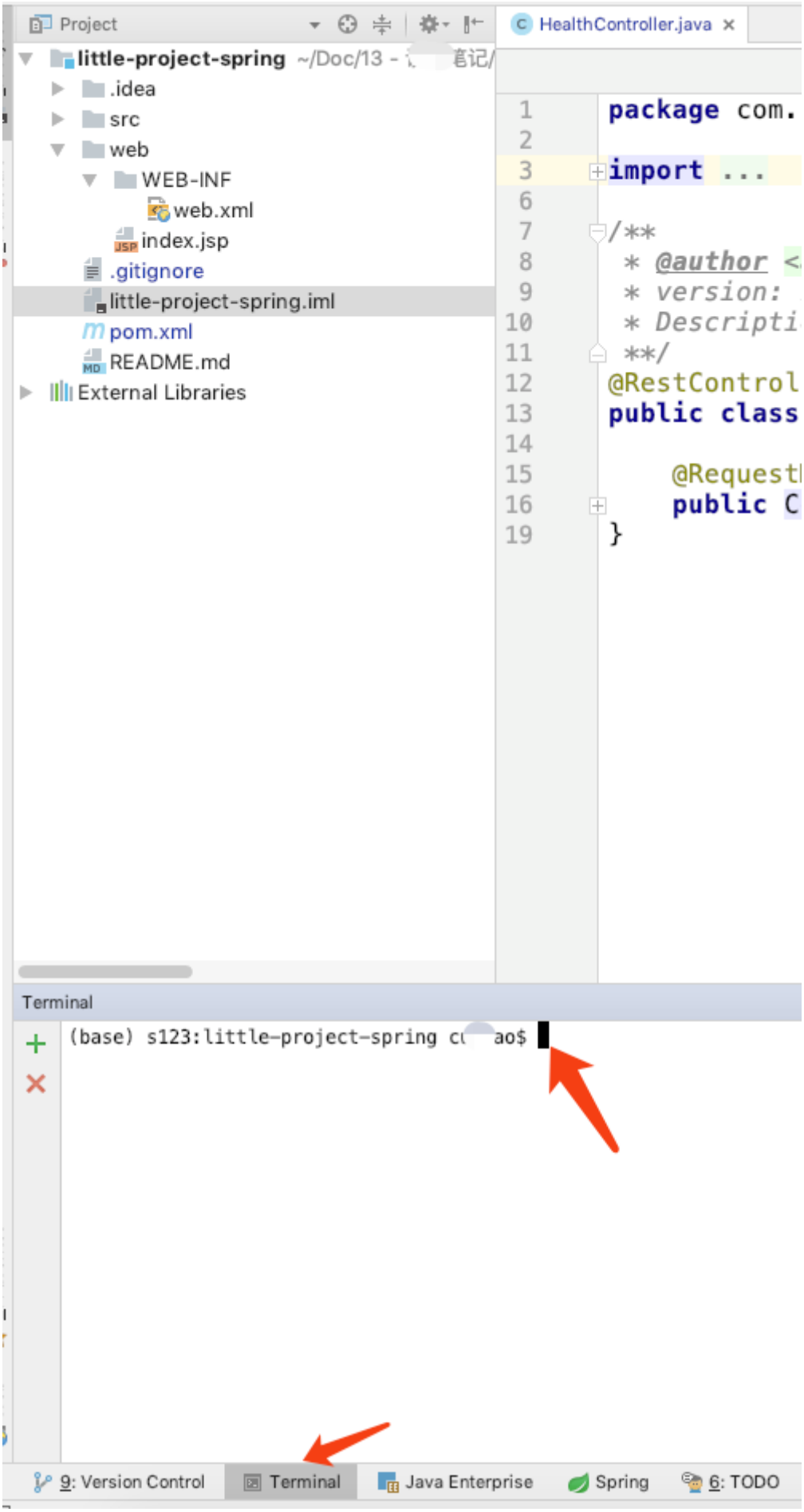
Project
little-project-spring ~/Doc/13 -
.idea
src
web
WEB-INF
web.xml
index.jsp
.gitignore
little-project-spring.iml
pom.xml
README.md
External Libraries
HealthController.java
package com.
import ...
/**
* @author
* version:
* Descripti
**/
@RestControl
public class
@Request
public C
}
Terminal
(base) s123:little-project-spring
ao$
9: Version Control
Terminal
Java Enterprise
Spring
6: TODO

图 2 打开命令行

如图 3 所示，这里输入“mvn clean package” 的命令，这里表明通过 Maven 进行打包，打包的是生产环境（prod）。

图 3 输入打包命令

在输入命令以后回车，如图 4 所示，会在输出信息中看到“BUILD SUCCESS”的字样表示，打包成功。同时在项目文件的 target 目录下面会看到一个“little-project-spring.war”的文件，这就是我们将要发布的 war 包。后面的操作会用到它。

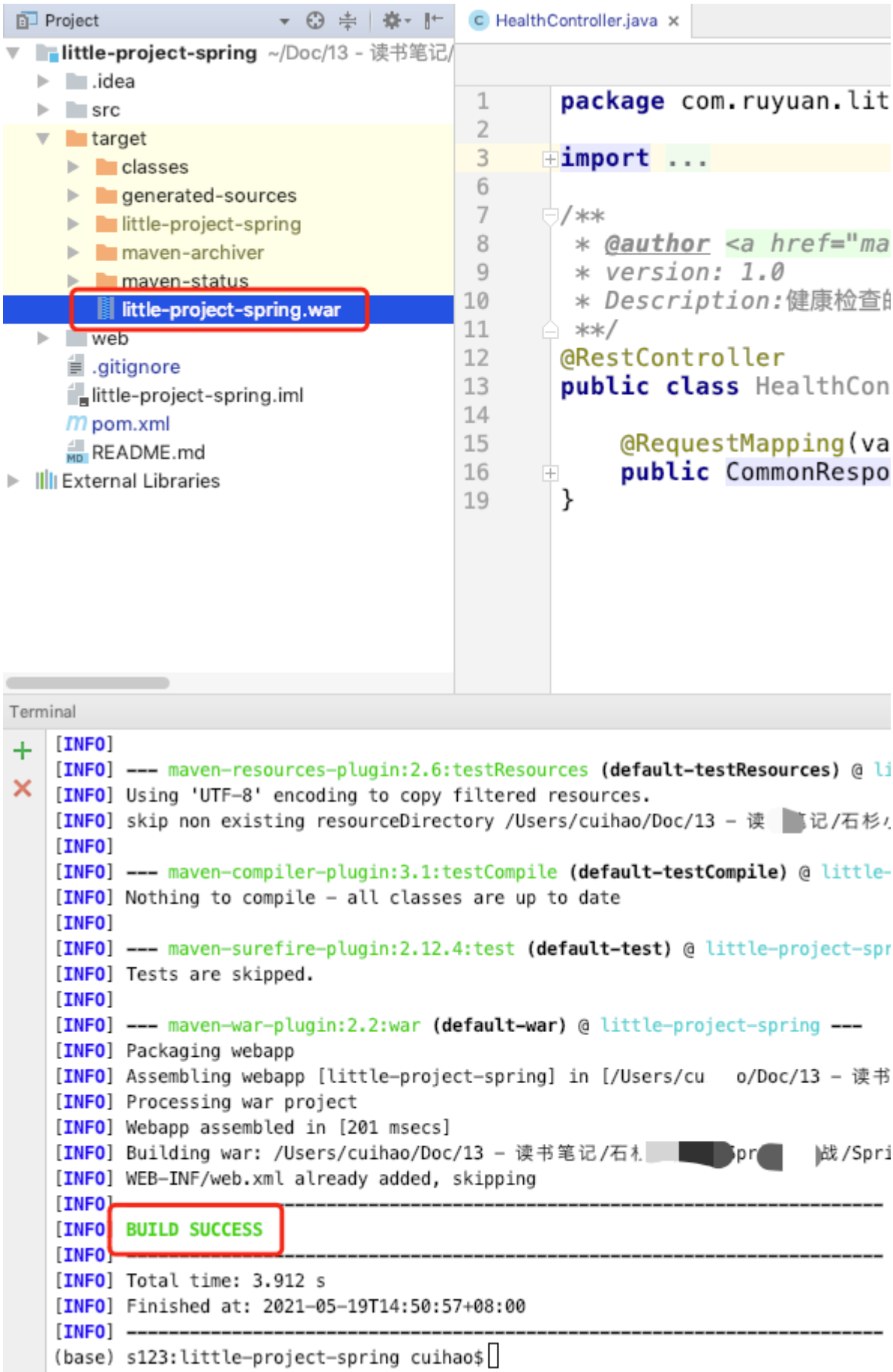
Project
little-project-spring ~/Doc/13 - 读书笔记/
.idea
src
target
classes
generated-sources
little-project-spring
maven-archiver
maven-status
little-project-spring.war
web
.gitignore
little-project-spring.iml
pom.xml
README.md
External Libraries
HealthController.java
1 package com.ruyuan.lit
2
3 import ...
6
7 /**
8 * @author <a href="ma
9 * version: 1.0
10 * Description:健康检查
11 **/
12 @RestController
13 public class HealthCon
14
15 @RequestMapping(va
16 public CommonRespo
19 }
Terminal
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ li
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/cuihao/Doc/13 - 读
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ little-
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ little-project-spr
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ little-project-spring ---
[INFO] Packaging webapp
[INFO] Assembling webapp [little-project-spring] in [/Users/cu o/Doc/13 - 读书
[INFO] Processing war project
[INFO] Webapp assembled in [201 msecs]
[INFO] Building war: /Users/cuihao/Doc/13 - 读书笔记/石
[INFO] WEB-INF/web.xml already added, skipping
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 3.912 s
[INFO] Finished at: 2021-05-19T14:50:57+08:00
[INFO] --
(base) s123:little-project-spring cuihao$

图 4 打包成功

3、上传服务

在上传服务（jar 包）之前需要保证，在对应的 ECS 中建立服务运行的目录如下：

/home/admin/little-project-spring/

这个目录已经由儒猿团队在 ECS 的镜像中创建好了，大家不用手动创建，我们后面的发布就基于这个目录。其中“/home/admin/little-project-spring”是用来存放部署脚本的，注意我们这里使用的 deploy.sh 脚本已经由儒猿团队上传了。

“/home/admin/little-project-spring/”目录是存放服务的 war 包的，这个包是需要我们自己上传的。

在 ECS 上建立好目录结构以后，再回到本地的项目中，依旧是在 IntelliJ IDEA 的命令行中输入以下命令：

```
scp target/little-project-spring.war  root@47.117.120.102:/home/admin/little-project-spring /
```

命令的意思是通过 scp 命令将刚才打包的“little-project-spring.war”文件 copy 到对应 ECS 服务器的“/home/admin/little-project- spring/”目录中。这里的“47.117.120.102”是我的测试地址，大家可以更换为自己申请的 ECS 的 IP 地址。

这里需要特别说明一下，由于我在执行命令的根目录在“little-project-spring”项目下面，如果你在其他的地方执行上述两条命令，需要指定好源文件的目录。同时在使用 scp 命令以后会让大家输入服务器的密码，该密码可以从实战云平台上获取。

完成上面两个命令以后，服务就已经部署到 ECS 了。

4、启动服务

完成部署以后，需要到 ECS 服务器上面启动部署的酒店管理后台的服务。还是登录 ECS 服务器，通过以下命令进入到“deploy.sh”文件所在的目录。

```
cd /home/admin/little-project-spring
```

deploy.sh 文件是儒猿团队为大家生成的发布的脚本文件，通过 Linux shell 脚本完成发布的参数配置和启动命令。包括启动应用等待的时间、应用端口号、健康检查的 URL 以及 jar 包的目录和日志信息。有兴趣的同学可以打开看看，这里就不做展开的介绍了。

保证当前目录下面存在“deploy.sh”文件，使用如下命令启动服务。

```
sh deploy.sh restart
```

命令使用了“restart”作用与“start”是一致的，用“restart”的目的以免在重复发布过程中，学员忘记是否启动过服务。因此使用“restart”，这样即便是已经启动过服务，也会重新加载服务。

运行命令在看到图 5 所示的“started java process”字样的时候，就说明服务启动成功了。但是随后的健康检查服务一直在重试，由于我们在 health 方法里面强行抛出了 exception 导致调用这个方法的时候报错，返回 500 错误。

```
[root@iZuf69caqdflvzby0x0dlvZ little-project-spring]# sh deploy.sh restart
stop java process
        -- stopping java lasts 59 seconds.
java process has exited

starting java process
Using CATALINA_BASE:   /home/admin/little-project-spring/apache-tomcat-9.0.45
Using CATALINA_HOME:   /home/admin/little-project-spring/apache-tomcat-9.0.45
Using CATALINA_TMPDIR: /home/admin/little-project-spring/apache-tomcat-9.0.45/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /home/admin/little-project-spring/apache-tomcat-9.0.45/bin/bootstrap.ja
r:/home/admin/little-project-spring/apache-tomcat-9.0.45/bin/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
started java process
checking http://127.0.0.1:8090
Wait app to pass health check: 1...
code is 500
Wait app to pass health check: 2...
code is 500
Wait app to pass health check: 3...
```

图 4 启动服务成功。

如图 5 所示，同样可以通过浏览器输入 IP 地址加上端口 8090 来查看健康检查服务是否工作。http://47.117.115.80:8090，其中把 IP 地址换成你自己的。可以看到返回 500 的错误信息，比起之前的错误来说，这个错误提示“系统异常，请联系

管理员”。显然将这个错误信息暴露对用户是比较友好的，这都得益于我们使用了统一的错误处理类，在实际工作中可以将错误都导向同一个错误页面，然后再把错误信息打印到页面上。

{"code":500,"message":"系统异常,请联系管理员","data":null}

图 5 访问健康服务返回友好的错误信息

5、总结

这节课验证上节课编写的代码，通过 GlobalExceptionHandler 类处理 Controller 中抛出的异常信息，从而解决页面报错的问题。下节课通过使用 Spring AOP 的方式拦截登陆信息，并且判断请求信息是否合乎系统要求。下期见，拜拜。