Spring Boot 第二章异常处理与单元测试

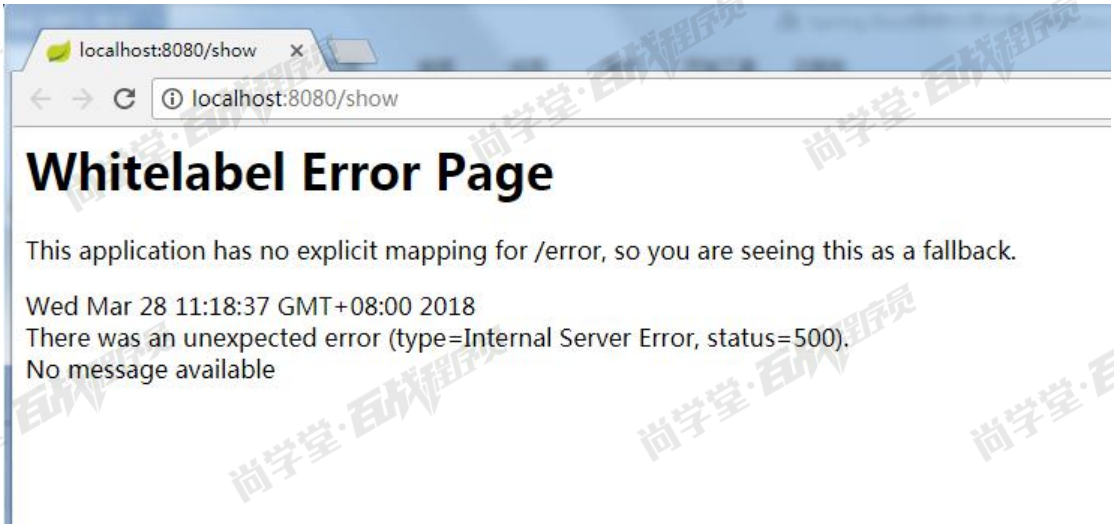
（SpringBoot 高级）

一、 SpringBoot 中异常处理方式

1 SpringBoot 中对于异常处理提供了五种处理方式

1.1 自定义错误页面

SpringBoot 默认的处理异常的机制：SpringBoot 默认的已经提供了一套处理异常的机制。一旦程序中出现了异常 SpringBoot 会像/error 的 url 发送请求。在 springBoot 中提供了一个叫 BasicExceptionController 来处理/error 请求，然后跳转到默认显示异常的页面来展示异常信息。



如果我们需要将所有的异常同一跳转到自定义的错误页面，需要再 src/main/resources/templates 目录下创建 error.html 页面。注意：名称必须叫 error

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>错误提示页面</title>
</head>
<body>
    出错了，请与管理员联系。。。
    <span th:text="${exception}"></span>
</body>
</html>
```

1.2 @ExceptionHandle 注解处理异常

1.2.1 Controller

```
/**
 * SpringBoot 处理异常方式一：自定义错误页面
 *
 *
 */
@Controller
public class DemoController {

    @RequestMapping("/show")
    public String showInfo(){
        String str = null;
        str.length();
        return "index";
    }

    @RequestMapping("/show2")
    public String showInfo2(){
        int a = 10/0;
        return "index";
    }

    /**
     * java.lang.ArithmeticException
     * 该方法需要返回一个 ModelAndView：目的是可以让我们封装异常信息以及视图的指定
     * 参数 Exception e:会将产生异常对象注入到方法中
     */
    @ExceptionHandler(value={java.lang.ArithmeticException.class})
    public ModelAndView arithmeticExceptionHandler(Exception e){
        ModelAndView mv = new ModelAndView();
        mv.addObject("error", e.toString());
        mv.setViewName("error1");
        return mv;
    }

    /**
     * java.lang.NullPointerException
     * 该方法需要返回一个 ModelAndView：目的是可以让我们封装异常信息以及视图的指定
```

```
     * 参数 Exception e:会将产生异常对象注入到方法中
     */
    @ExceptionHandler(value={java.lang.NullPointerException.class})
    public ModelAndView nullPointerExceptionHandler(Exception e){
        ModelAndView mv = new ModelAndView();
        mv.addObject("error", e.toString());
        mv.setViewName("error2");
        return mv;
    }
}
```

1.2.2 页面

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>错误提示页面-ArithmeticException</title>
</head>
<body>
    出错了，请与管理员联系。。。
    <span th:text="${error}"></span>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>错误提示页面-NullPointerException</title>
</head>
<body>
    出错了，请与管理员联系。。。
    <span th:text="${error}"></span>
</body>
</html>
```

1.3 @ControllerAdvice+@ExceptionHandler 注解处理异常

1.3.1 需要创建一个能够处理异常的全局异常类。在该类上需要添加@ControllerAdvice 注解

```
/**
 * 全局异常处理类
 *
 *
 */
@ControllerAdvice
public class GlobalException {
    /**
     * java.lang.ArithmeticException
     * 该方法需要返回一个 ModelAndView：目的是可以让我们封装异常信息以及视图的指定
     * 参数 Exception e:会将产生异常对象注入到方法中
     */
    @ExceptionHandler(value={java.lang.ArithmeticException.class})
    public ModelAndView arithmeticExceptionHandler(Exception e){
        ModelAndView mv = new ModelAndView();
        mv.addObject("error", e.toString());
        mv.setViewName("error1");
        return mv;
    }

    /**
     * java.lang.NullPointerException
     * 该方法需要返回一个 ModelAndView：目的是可以让我们封装异常信息以及视图的指定
     * 参数 Exception e:会将产生异常对象注入到方法中
     */
    @ExceptionHandler(value={java.lang.NullPointerException.class})
    public ModelAndView nullPointerExceptionHandler(Exception e){
        ModelAndView mv = new ModelAndView();
        mv.addObject("error", e.toString());
        mv.setViewName("error2");
        return mv;
    }

}
```

1.4配置 SimpleMappingExceptionResolver 处理异常

1.4.1 在全局异常类中添加一个方法完成异常的同一处理

```
    /**
     * 通过 SimpleMappingExceptionResolver 做全局异常处理
     *
     *
     */
    @Configuration
    public class GlobalException {

        /**
         * 该方法必须要有返回值。返回值类型必须是:
SimpleMappingExceptionResolver
         */
        @Bean
        public SimpleMappingExceptionResolver
getSimpleMappingExceptionResolver(){
            SimpleMappingExceptionResolver resolver = new
SimpleMappingExceptionResolver();

            Properties mappings = new Properties();

            /**
             * 参数一：异常的类型，注意必须是异常类型的全名
             * 参数二：视图名称
             */
            mappings.put("java.lang.ArithmeticException", "error1");
            mappings.put("java.lang.NullPointerException","error2");

            //设置异常与视图映射信息的
            resolver.setExceptionMappings(mappings);

            return resolver;
        }

    }
```

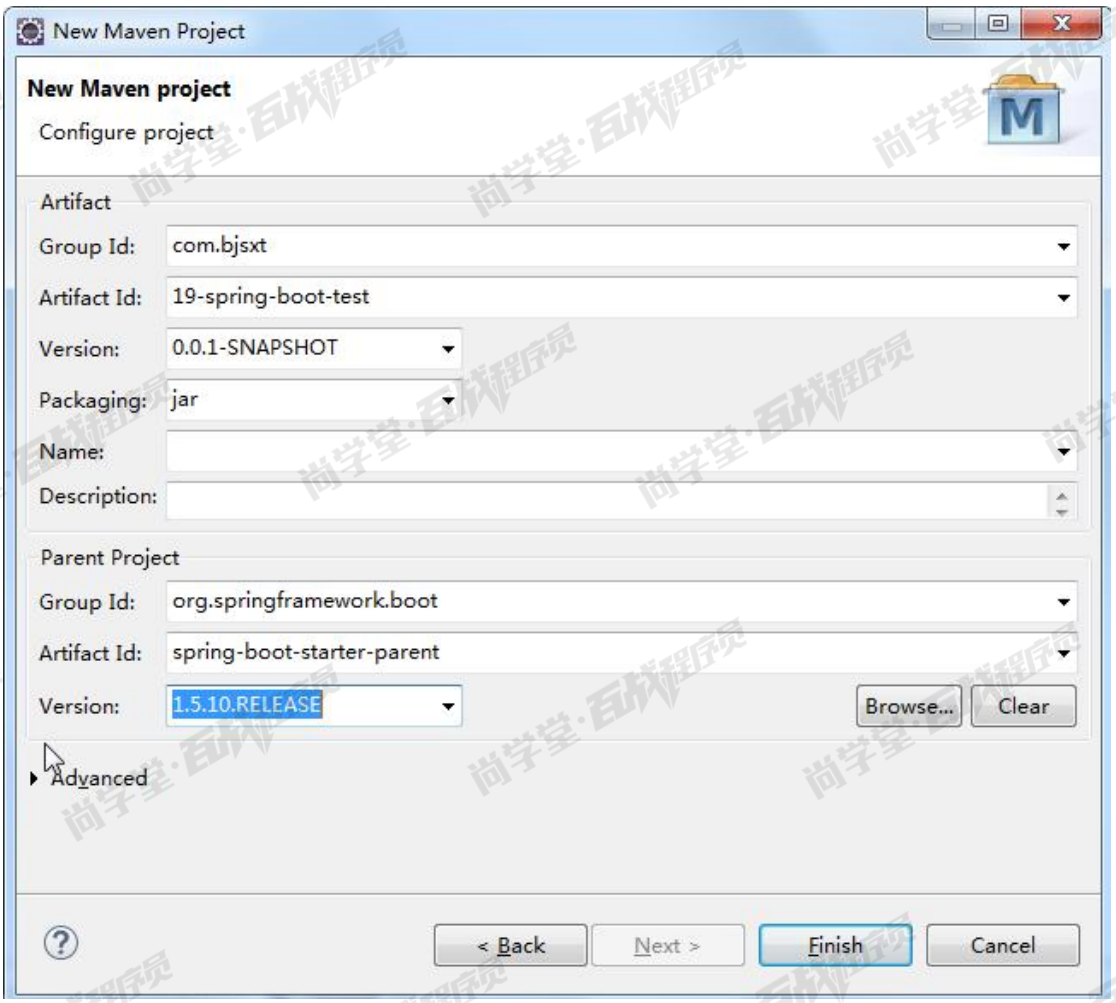
1.5 自定义 HandlerExceptionResolver 类处理异常

1.5.1 需要再全局异常处理类中实现 HandlerExceptionResolver 接口

```
/**
 * 通过实现 HandlerExceptionResolver 接口做全局异常处理
 *
 *
 */
@Configuration
public class GlobalException implements HandlerExceptionResolver {

    @Override
    public ModelAndView resolveException(HttpServletRequest request,
HttpServletResponse response, Object handler,
            Exception ex) {
        ModelAndView mv = new ModelAndView();
        //判断不同异常类型，做不同视图跳转
        if(ex instanceof ArithmeticException){
            mv.setViewName("error1");
        }

        if(ex instanceof NullPointerException){
            mv.setViewName("error2");
        }
        mv.addObject("error", ex.toString());

        return mv;
    }
}
```

二、 Spring Boot 整合 Junit 单元测试

1 创建项目



2 修改 pom 文件

```
    <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.10.RELEASE</version>
      </parent>
      <groupId>com.bjsxt</groupId>
      <artifactId>19-spring-boot-test</artifactId>
      <version>0.0.1-SNAPSHOT</version>

      <properties>
```

```
    <java.version>1.7</java.version>
  </properties>

  <dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- 添加 junit 环境的 jar 包 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-test</artifactId>
    </dependency>
  </dependencies>
</project>
```

3 编写业务代码

3.1 Dao

```
@Repository
public class UserDaoImpl {

    public void saveUser(){
        System.out.println("insert into users.....");
    }
}
```

3.2 业务层

```
@Service
public class UserServiceImpl {

    @Autowired
    private UserDaoImpl userDaoImpl;

    public void addUser(){
        this.userDaoImpl.saveUser();
    }
}
```

3.3编写启动类

```
@SpringBootApplication
public class App {

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }

}
```

4 使用 SpringBoot 整合 Junit 做单元测试

4.1编写测试类

```
/**
 * SpringBoot 测试类
 *@RunWith:启动器
 *SpringJUnit4ClassRunner.class：让 junit 与 spring 环境进行整合
 *
 *@SpringBootTest(classes={App.class}) 1,当前类为 springBoot 的测试类
 *@SpringBootTest(classes={App.class}) 2,加载 SpringBoot 启动类。启动
springBoot
 *
 *junit 与 spring 整合
@Contextconfiguartion("classpath:applicationContext.xml")
 */
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes={App.class})
public class UserServiceTest {

    @Autowired
    private UserServiceImpl userServiceImpl;

    @Test
    public void testAddUser(){
        this.userServiceImpl.addUser();
    }
}
```