1 基本流程

1.1 创建项目

1. IDEA

Spring Initializr

2. Web

https://start.spring.io/

1.2 运行项目

1. IDEA

直接运行 XxxApplication 中的 main 方法。

2. Maven

在项目文件夹中运行 mvn spring-boot:run

3. java

在项目文件夹中运行 mvn install 进行编译。

进入 target 文件夹，使用 java -jar projectname.jar 运行。

1.3 访问项目

http://127.0.0.1:8080/

2 项目配置

2.1 配置文件

默认使用 application.properties，也可以使用 .yml，语法更简洁。

yml 的值与键的冒号之间必须有空格。

2.2 常用配置

```
server:
  port: 8080
```

```
  context-path: /summary
```

2.3 配置中使用配置

```
configItem: value
content: "configItem: ${configItem}"
```

2.4 获取配置

```
@Value("${configItem}")
private String configItem;
```

2.5 批量获取配置

application.yml

```
configs:
  key1: value1
  key2: value2
```

ConfigsProperties.java

```
@Component
@ConfigurationProperties(prefix = "user")
public class ConfigsProperties {
    private String name;
    private int age;
    //getter setter
}
```

IndexController.java

```
@RestController
public class IndexController {
    @Autowired
    private ConfigsProperties configsProperties;
    @RequestMapping(value = "/hello", method =
RequestMethod.GET)
    public String say() {
        return configsProperties.getName();
    }
}
```

2.6 配置切换

将配置写入 application-dev.yml 和 application-prod.yml，在 application.yml 中配置如下项：

```
spring:
  profiles:
      active: dev
```

application.yml 中为共用配置。

或在运行时指定配置文件：

```
java -jar projectname.jar --spring. profiles. active=dev
```

3 Controller 的使用

3.1 Controller 常用注解

注解	功能
@Controller	处理 http 请求
@RestController	Spring4 新增，原返回 json 需@ResponseBody 配合 @Controller
@RequestMapping	配置 url 配置

注解	功能
@PathVariable	获取 url 的数据
@RequestParam	获取请求参数的值，value，required，defaultValue
@GetMapping @PostMapping	组合注解

4 数据库操作

4.1 Spring-Data-Jpa

JPA(java persistence API)定义了一系列对象持久化的标准。

4.2 依赖组件

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

4.3 使用 Repository

继承 JpaRepository，基本操作不需要添加任何代码。

```
public interface NewsRepository extends JpaRepository<News,
Integer>{}
```

此时可以使用以下方法：

```
findAll   //查找全部数据
save      //没有主键时为增加、有主键时更新
findOne   //使用主键查找一条记录
delete    //删除
```

若需要增加其他查找方法，可在 Repository 接口中增加：

```
public List<News> findByTitle(String title);
```

方法命名必须严格按照标准，不需要实现即可直接调用。

4.4 事务注解

```
/* 添加两条新闻，使用事务，即两条新闻要么都插入成功，要么都不插入*/
@Transactional
public void insertTwoNews(){}
```

5 表单验证

1. 在 Model 中为需要验证的字段添加注解，如：

```
@Min(value = 18, messsage = ”Exception Message”)
```

```
@NotEmpty(message = ”姓名必填”)
private String name;
```

2. 在 Controller 的接收参数中为需要验证的字段添加注解，如：

```
@Valid ModelClass model, BindingResult bindingResult
```

```
public ResultVO<Map<String, String>> create(@Valid OrderForm
orderForm, BindingResult bindingResult) {…}
```

3. 在对应的处理方法中即可使用，如：

```
if(bindingResult.hasErrors()){
  //输出 bindingResult.getFieldError().getDefaultMessage()
  return null;
}
```

5.1 自定义校验

5.1.1 创建校验注解

设置使用目标、设置运行时、设置校验类、添加三个必要属性

```
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = MyConstraintValidator.class)
public @interface MyConstraint {

    String message() default
"{org.hibernate.validator.constraints.NotBlank.message}";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };

}
```

5.1.2 创建校验类

实现 ConstraintValidator，指定注解和针对的类型、实现初始化方法、实现判别方法

```
public class MyConstraintValidator implements
ConstraintValidator<MyConstraint, Object> {
    @Override
    public void initialize(MyConstraint constraintAnnotation) {

    }

    @Override
```

```
    public boolean isValid(Object value,
ConstraintValidatorContext context) {
        return false;
    }
}
```

5.1.3 说明

校验类实现了接口后，隐式添减注解为 Bean

校验类可以直接使用@Autowire 资源

6 AOP

6.1 目的

将通用逻辑从业务逻辑中分离出来

6.2 依赖

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

6.3 创建切面类和方法

6.3.1 基本

```
@Aspect
@Component
public class HttpAspect {
    @Before("execution(public *
cn.iecas.mysql.controller.NewsController.*(..))")
    public void doBefore() {
        System.out.println("接口调用之前");
    }
    @After("execution(public *
cn.iecas.mysql.controller.NewsController.*(..))")
    public void doAfter() {
        System.out.println("接口调用之后");
    }
}
```

6.3.2 改进

```
@Aspect
@Component
public class HttpAspect {
    private final static Logger logger=
LoggerFactory.getLogger(HttpAspect.class);
    @Pointcut("execution(public *
cn.iecas.mysql.controller.NewsController.*(..))")
    public void log() {
    }
    @Before("log()")
    public void doBefore() {
        logger.info("接口调用之前");
    }
    @After("log()")
    public void doAfter() {
        logger.info("接口调用之后");
    }
}
```

6.3.3拓展

6.3.3.1 请求信息获取

```
@Before("log()")
public void doBefore(JoinPoint joinPoint) {
    ServletRequestAttributes attributes =
(ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
    HttpServletRequest request = attributes.getRequest();
    //url
    logger.info("url={}", request.getRequestURI());
    //method
    logger.info("method={}", request.getMethod());
    //ip
    logger.info("ip={}", request.getRemoteAddr());
    //类方法
    logger.info("class_method={}",
joinPoint.getSignature().getDeclaringTypeName()
            + "."
            + joinPoint.getSignature().getName());
    //参数
    logger.info("args={}", joinPoint.getArgs());
}
```

6.3.3.2 返回拦截

```
@AfterReturning(returning = "object", pointcut = "log()")
public void doAfterReturning(Object object) {
    logger.info("response={}", object.toString());
}
```

7 异常处理

7.1 统一结果格式

Result：Integer code、String msg、T date

ResultUtil：success(obj)、error(code, msg)

7.2 异常信息管理

```
public enum ResultEnum {
    UNKNOW_ERROR(-1, "未知错误"),
    SUCCESS(0, "成功");
    private Integer code;
    private String msg;
    ResultEnum(Integer code, String msg) {
        this.code = code;
        this.msg = msg;
    }
    public Integer getCode() {
        return code;
    }
    public String getMsg() {
        return msg;
    }
}
```

7.3 自定义异常

```
public class RainException extends RuntimeException {
    private Integer code;
    public RainException(ResultEnum resultEnum) {
        super(resultEnum.getMsg());
        this.code = resultEnum.getCode();
    }
    public Integer getCode() {
```

```
        return code;
    }
    public void setCode(Integer code) {
        this.code = code;
    }
}
```

7.4 自定义异常处理

```
@ControllerAdvice
public class ExceptionHandle {
    private final static Logger logger =
LoggerFactory.getLogger(ExceptionHandle.class);
    @ExceptionHandler(value = Exception.class)//需要处理的异常类型
    @ResponseBody
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public Result handle(Exception e) {
        if (e instanceof RainException) {
            RainException rainException = (RainException) e;
            return ResultUtil.error(rainException.getCode(),
rainException.getMessage());
        } else {
            logger.error("系统异常: ", e);
            return ResultUtil.error(-1, "未知错误");
        }
    }
}
```

8 单元测试

8.1 普通测试

1. 添加类注解

```
@RunWith(SpringRunner.class)
@SpringBootTest
```

2. 添加方法注解

```
@Test
public void listPages() throws Exception {
}
```

3. 方法调用

```
List<Integer> pageList = PageTool.listPages(1, 100, 5);
```

4. 添加断言

```
Assert.assertEquals(new Integer(1), pageList.get(0));
Assert.assertNotNull(object);
Assert.assertNotEquals(unexpected,actual);
```

8.2 Web 接口测试

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class NewsControllerTest {
    @Autowired
    MockMvc mockMvc;
    @Test
    public void listNews() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/news"))
                .andExpect(MockMvcResultMatchers.status().isOk())
;
    }
}
```

8.3 测试注解

8.3.1 @Transactional 注解

在测试中使用表示完成后回滚数据库，防止数据库污染。

9 参考资料

慕课网

《Spring Boot 实战》