**springboot学习**

# 新建springboot项目

## 启动的三种方式

1. 在eclipse中Application类中右键使用java application方式启动

2. 在cmd下切换到项目目录下，使用 mvn spring-boot:run

3. 在cmd下切换到项目目录下，使用mvn clean package将项目打包，打包后的jar在./target目录下，再使用java来启动：java -jar test-0.0.1-SNAPSHOT.jar

## 配置

尽量使用application.yml

|  |
| --- |
| server:  port: 8081  servlet:  context-path: /luckymoney  minMoney: 1 # “:”后面要加空格，不然不符合yml的格式要求  description: 最少发${minMoney}元 #配置中也可使用其它配置项定义的值 |

### 使用对象配置

|  |
| --- |
| limit:  minMoney: 1  maxMoney: 100  description: 最少发${limit.minMoney}元，最多发${limit.maxMoney}元 |
| @Component  @ConfigurationProperties(prefix="limit")  **public** **class** LimitConfig {  **private** BigDecimal minMoney;  **private** BigDecimal maxMoney;  **private** String description; |

### 区分开发和生产环境

新增application-dev.yml和application-prod.yml两个配置文件分别存放开发和生产环境需要的配置

在application.yml中添加配置

|  |
| --- |
| spring:  profiles:  active: dev #dev表示当前应用开发环境 |

使用dev环境jar包启动生产环境

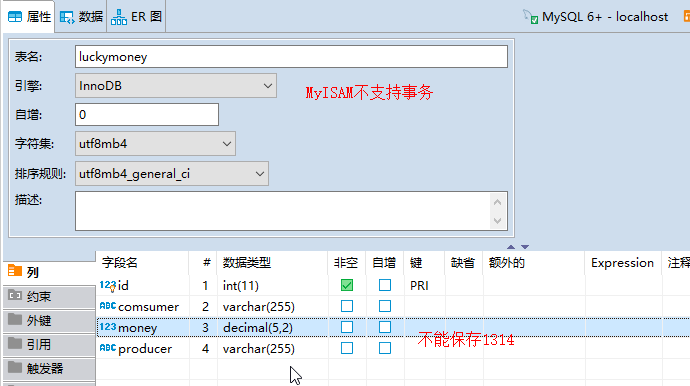
java -jar -Dspring.profiles.active=prod test-0.0.1-SNAPSHOT.jar

## 增查改示例

|  |
| --- |
| @RestController  **public** **class** LuckymoneyController {  @Autowired  **private** LuckymoneyDao luckymoneyDao;  @Autowired  **private** LuckymoneyService luckymoneyService;  /\*\*  \* 获取红包列表  \* **@return**  \*/  @GetMapping("/luckymoneys")  **public** List<Luckymoney> list(){  List<Luckymoney> list = luckymoneyDao.findAll();  **return** list;  }  /\*\*  \* 创建红包  \* **@return**  \*/  @PostMapping("/luckymoneys")  **public** Luckymoney create(@RequestParam("producer") String producer,  @RequestParam("money") BigDecimal money,  @RequestParam("comsumer") String comsumer) {  Luckymoney luckymoney = **new** Luckymoney();  luckymoney.setMoney(money);  luckymoney.setComsumer(comsumer);  luckymoney.setProducer(producer);  Luckymoney result = luckymoneyDao.save(luckymoney);  **return** result;  }  /\*\*  \* 通过id查询红包信息  \* **@param** id  \* **@return**  \*/  @GetMapping("/luckymoneys/{id}")  **public** Luckymoney findById(@PathVariable Integer id) {  **return** luckymoneyDao.findById(id).orElse(**null**);  }  @PutMapping("/luckymoneys/{id}")  **public** Luckymoney update(@PathVariable Integer id,  @RequestParam("consumer") String consumer) {  Luckymoney luckymoney = findById(id);  **if**(luckymoney != **null**) {  luckymoney.setComsumer(consumer);  Luckymoney result = luckymoneyDao.save(luckymoney);  **return** result;  }  **return** **null**;  }  @PostMapping("/luckymoneys/two")  **public** **void** saveTwo() {  luckymoneyService.saveTwo();  }  } |

### 事务示例

|  |
| --- |
| @Service  **public** **class** LuckymoneyService {  @Autowired  **private** LuckymoneyDao luckymoneyDao;    @Transactional //添加事务，方法中出现异常，所有事务操作都回滚  **public** **void** saveTwo() {  Luckymoney luckymoney = **new** Luckymoney();  luckymoney.setMoney(**new** BigDecimal(520));  luckymoney.setProducer("me");  luckymoney.setComsumer("her");  luckymoneyDao.save(luckymoney);    Luckymoney luckymoney2 = **new** Luckymoney();  luckymoney2.setMoney(**new** BigDecimal(1314));  luckymoney2.setProducer("me");  luckymoney2.setComsumer("her");  luckymoneyDao.save(luckymoney2);  }  } |



## 日志示例

|  |
| --- |
| @Aspect  @Component  **public** **class** HttpAspect {    **private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(HttpAspect.**class**);    @Pointcut("execution(public \* com.imooc.controller.GirlController.\*(..))") //定义切面  **public** **void** pointcut1() {}    @Before("pointcut1()") //在切面方法执行前执行  **public** **void** log(JoinPoint joinPoint) {  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = attributes.getRequest();  ***log***.info("url={}",request.getRequestURL()); //请求url  ***log***.info("method={}",request.getMethod()); //请求方法  ***log***.info("ip={}",request.getRemoteAddr()); //客户端ip  //类名 和 方法  ***log***.info("class\_method={}",joinPoint.getSignature().getDeclaringTypeName()+"."+joinPoint.getSignature().getName());  ***log***.info("args={}",joinPoint.getArgs()); //参数  }    @After("pointcut1()") //在切面方法执行后执行  **public** **void** after() {  ***log***.info("after.......");  }    @AfterReturning(returning="object", pointcut="pointcut1()") //在切面方法执行后，结果返回时执行  **public** **void** afterReturning(Object object) {  ***log***.info("response={}",object.toString());  }  } |

## 通用返回对象封装示例

|  |
| --- |
| **public** **class** Result <T> {  **private** Integer code; //返回码  **private** String msg; //提示信息  **private** T data; //成功时返回的内容    **public** Result() {    }  **public** Result(Integer code, String msg, T data) {  **this**.code = code;  **this**.msg = msg;  **this**.data = data;  }  //getter... setter... toString...    **public** **static** Result ok() {  **return** *ok*(**null**);  }  **public** **static** Result ok(Object object) {  **return** **new** Result(ResultEnums.***SUCCESS***.getCode(),ResultEnums.***SUCCESS***.getMsg(), object);  }  **public** **static** Result build(ResultEnums enums) {  **return** **new** Result(enums.getCode(), enums.getMsg() ,**null**);  }  **public** **static** Result build(Integer code, String msg) {  **return** **new** Result(code, msg ,**null**);  }    } |

## 通用异常类定义

用于handler捕获异常，返回特定格式信息。

必须继承RuntimeException，否则service层抛出非运行时异常不能回滚事务

|  |
| --- |
| **public** **class** GirlException **extends** RuntimeException {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Integer code;  **public** GirlException(ResultEnums enums) {  **super**(enums.getMsg());  **this**.code = enums.getCode();  }  **public** Integer getCode() {  **return** code;  }  **public** **void** setCode(Integer code) {  **this**.code = code;  }  @Override  **public** String toString() {  **return** "GirlException [code=" + code + ", getMessage()=" + getMessage() + "]";  }    } |

## 统一处理异常handler

分别捕获处理自定义异常，以及其它异常，使返回前台json数据格式一致。

|  |
| --- |
| @ControllerAdvice  **public** **class** ExceptionHandle {    **private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(ExceptionHandle.**class**);  @ExceptionHandler(value=Exception.**class**)  @ResponseBody  **public** Result handler(Exception e) {  **if**(e **instanceof** GirlException) {  GirlException girlException = (GirlException) e;  **return** Result.*build*(girlException.getCode(), girlException.getMessage());  } **else** {  ***log***.error("【系统异常】：", e);  **return** Result.*build*(ResultEnums.***UNKNOW\_ERROR***);  }  }  } |

## 通用返回结果枚举类型封装对象

|  |
| --- |
| **public** **enum** ResultEnums {  ***UNKNOW\_ERROR***(-1,"未知错误"),  ***SUCCESS***(0,"成功"),  ***PRIMARY\_SCHOOL***(100,"你可能还在上小学"),  ***MIDDLE\_SCHOOL***(200,"你应该在上初中");    **private** Integer code;  **private** String msg;  **private** ResultEnums(Integer code,String msg) {  **this**.code = code;  **this**.msg = msg;  }  **public** Integer getCode() {  **return** code;  }  **public** String getMsg() {  **return** msg;  }    } |

## 单元测试

### service层单元测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  **public** **class** GirlServiceTest {    @Autowired  GirlService girlService;    @Test  **public** **void** test1() **throws** Exception {  Assert.*assertEquals*(**new** Integer(8), girlService.getOne(11).getAge());  }  } |

### controller层单元测试

|  |
| --- |
| @RunWith(SpringRunner.**class**)  @SpringBootTest  @AutoConfigureMockMvc  **public** **class** GirlControllerTest {  @Autowired  **private** MockMvc mockMvc;    @Test  **public** **void** test() **throws** Exception {  mockMvc.perform(MockMvcRequestBuilders.*get*("/girls"))  .andExpect(MockMvcResultMatchers.*status*().isOk()) .andExpect(MockMvcResultMatchers.*content*().string("aaaa"));  }  } |

# 常用注解

|  |
| --- |
| //@RestController //相当于@ResponseBody和@Controller两个注解，返回json，控制bean生成  @Controller //控制bean是否放入spring容器中  @ResponseBody //在此类中全部方法都返回json；也可定义在方法上（当前方法返回json）  @RequestMapping("/hello") //定义Rest风格访问路径全局，可以写成数组形式  **public** **class** HelloController {  @Autowired //spring提供注解：注入bean的对象 byType(类名）,或使用@Qualifier("userDao") byName  //@Resource(name="") //j2EE提供注解：默认byName ，也可@Resource(type=Controller.class)指定byType  **private** LimitConfig limitConfig;    @GetMapping("/say/{id}") //@RequestMapping(value="/hello",method=RequestMethod.GET) 的简写  //@RequestMapping("/say") //定义Rest风格访问路径(要加上全局路径/hello/hi)，可以写成数组形式  **public** String say(@PathVariable Integer id) { //获取路径中的参数  **return** "Hello World! description=" + limitConfig.getDescription();  }  @PostMapping("/say2") //@RequestMapping(value="/say2",method=RequestMethod.POST) 的简写  **public** String say2(@RequestParam(value="id",required=**false**,defaultValue="0") Integer myid) { //获取参数中的值  **return** "id="+myid;  }  } |
| @Entity //标识实体类  **public** **class** Luckymoney {  @Id //标识主键  @GeneratedValue //自增  **private** Integer id;  **private** BigDecimal money;  **private** String producer; //发送方  **private** String comsumer; //接收方 |