第二章 实现登录功能

1. 数据库设计
2. 明文密码两次MD5处理
3. JSR303参数检验+全局异常处理器
4. 分布式Session

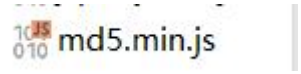
一、数据库设计

```
CREATE TABLE `miaosha_user` (
  `id` bigint(20) NOT NULL COMMENT '用户ID，手机号码',
  `nickname` varchar(255) NOT NULL,
  `password` varchar(32) DEFAULT NULL COMMENT 'MD5(MD5(pass明文+固定salt)+salt)',
  `salt` varchar(10) DEFAULT NULL,
  `head` varchar(128) DEFAULT NULL COMMENT '''头像''，云存储的ID',
  `register_date` datetime DEFAULT NULL COMMENT '注册时间',
  `last_login_date` datetime DEFAULT NULL COMMENT '上次登录时间',
  `login_count` int(11) DEFAULT '0' COMMENT '登录次数',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

二、明文密码二次MD5处理

MD5是一种不可逆的加密算法，若要验证密码是否相同，需要将输入的密码MD5加密得到密文然后与原先已经生成密文作对比。意思是有密文得不到明文，只能明文加密后做对比

第一次是页面js代码对用户输入的密码进行MD5加密




```
function doLogin(){
    g_showLoading();

    var inputPass = $("#password").val();
    var salt = g_passsword_salt;
    var str = ""+salt.charAt(0)+salt.charAt(2) + inputPass +salt.charAt(5) + salt.charAt(4)
    var password = md5(str);
```

第二次则是对客户端传来的处理过的密码再一次加密

```
String dbPass = user.getPassword();
String saltDB = user.getSalt();
String calcPass = MD5Util.formPassToDBPass(formPass, saltDB);
if(!calcPass.equals(dbPass)) {
    throw new GlobalException(CodeMsg.PASSWORD_ERROR);
}
```

这里做的是登录操作，如果是做注册操作则需要生成随机salt进行加密，并把盐存入到数据库中以备登录时进行验证。

三、JSR303参数校验以及全局异常处理

参数校验先添加依赖

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

然后加注解

```
@NotNull
@IsMobile
private String mobile;

@NotNull
@Length(min=32)
private String password;
```

isMoblie是自定义的注解

定义步骤：

接口：

```
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER })
@Retention(RUNTIME)
@Documented
@Constraint(validatedBy = {IsMobileValidator.class })
public @interface IsMobile {

    boolean required() default true;

    String message() default "手机号码格式错误";

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

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

```
public class IsMobileValidator implements ConstraintValidator<IsMobile, String> {

    private boolean required = false;

    public void initialize(IsMobile constraintAnnotation) { required = constraintAnnotation.re

    public boolean isValid(String value, ConstraintValidatorContext context) {
        if(required) {
            return ValidatorUtil.isMobile(value);
        }else {
            if(StringUtils.isEmpty(value)) {
                return true;
            }else {
                return ValidatorUtil.isMobile(value);
            }
        }
```

全局异常处理：

定义一个异常类： 也就是继承RuntimeException

```java
public class GlobalException extends RuntimeException{

    private static final long serialVersionUID = 1L;

    private CodeMsg cm;

    public GlobalException(CodeMsg cm) {
        super(cm.toString());
        this.cm = cm;
    }

    public CodeMsg getCm() { return cm; }

}
```

接下来写个切片 用途是拦截这些异常并且向客户端返回我们想要的错误信息格式

```java
@ControllerAdvice
@ResponseBody
public class GlobalExceptionHandler {
    @ExceptionHandler(value=Exception.class)
    public Result<String> exceptionHandler(HttpServletRequest request, Exception e){
        e.printStackTrace();
        if(e instanceof GlobalException) {
            GlobalException ex = (GlobalException)e;
            return Result.error(ex.getCm());
        }else if(e instanceof BindException) {
            BindException ex = (BindException)e;
            List<ObjectError> errors = ex.getAllErrors();
            ObjectError error = errors.get(0);
            String msg = error.getDefaultMessage();
            return Result.error(CodeMsg.BIND_ERROR.fillArgs(msg));
        }else {
            return Result.error(CodeMsg.SERVER_ERROR);
        }
    }
```

四、分布式session

其中一种实现方式：

主要是把登录的信息给保存到redis中，同时redis要做集群，redis中登录信息的key是token即一个uuid，value是登录信息，token存入cookies中。

```
public boolean login(HttpServletResponse response, LoginVo loginVo) {
    if(loginVo == null) {
        throw new GlobalException(CodeMsg.SERVER_ERROR);
    }
    String mobile = loginVo.getMobile();
    String formPass = loginVo.getPassword();
    //判断手机号是否存在
    MiaoshaUser user = getById(Long.parseLong(mobile));
    if(user == null) {
        throw new GlobalException(CodeMsg.MOBILE_NOT_EXIST);
    }
    //验证密码
    String dbPass = user.getPassword();
    String saltDB = user.getSalt();
    String calcPass = MD5Util.formPassToDBPass(formPass, saltDB);
    if(!calcPass.equals(dbPass)) {
        throw new GlobalException(CodeMsg.PASSWORD_ERROR);
    }
    //生成cookie

    addCookie(response, user);
    return true;
}


private void addCookie(HttpServletResponse response,MiaoshaUser user){
    String token = UUIDUtil.uuid();
    redisService.set(MiaoshaUserKey.token,token,user);
    Cookie cookie = new Cookie(COOKI_NAME_TOKEN,token);
    cookie.setMaxAge(MiaoshaUserKey.TOKEN_EXPIRE);
    cookie.setPath("/");
    response.addCookie(cookie);
}
```

取登录信息

```
public MiaoshaUser getByToken(HttpServletResponse response,String token) {
    if (StringUtils.isEmpty(token)){
        return null;

    }

    MiaoshaUser user = redisService.get(MiaoshaUserKey.token, token, MiaoshaUser.class);
    if (user != null){
        addCookie(response,user);
    }
    return user;

}
```

为了让秒杀商品时写业务每次都要重复写cookie的获取与用户信息的获取，将这些逻辑写入UserArgumentResolver中， 该类实现了HandlerMethodArgumentResolver接口，并且将这个类的实例加入到argumentResolvers这个list中，通过这一系列操作后，springmvc可以将页面传来的数据做出处理并且自动封装MiaoshaUser，

具体实现：

```java
@Service
public class UserArgumentResolver implements HandlerMethodArgumentResolver {

  @Autowired
  MiaoshaUserService userService;

  public boolean supportsParameter(MethodParameter parameter) {
    Class<?> clazz = parameter.getParameterType();
    return clazz==MiaoshaUser.class;
  }

  public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer
mavContainer,
      NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception {
    HttpServletRequest request = webRequest.getNativeRequest(HttpServletRequest.class);
    HttpServletResponse response = webRequest.getNativeResponse(HttpServletResponse.class);

    String paramToken = request.getParameter(MiaoshaUserService.COOKI_NAME_TOKEN);
    String cookieToken = getCookieValue(request, MiaoshaUserService.COOKI_NAME_TOKEN);
    if(StringUtils.isEmpty(cookieToken) && StringUtils.isEmpty(paramToken)) {
      return null;
    }
    String token = StringUtils.isEmpty(paramToken)?cookieToken:paramToken;
    return userService.getByToken(response, token);
  }

  private String getCookieValue(HttpServletRequest request, String cookiName) {
    Cookie[]  cookies = request.getCookies();
    for(Cookie cookie : cookies) {
      if(cookie.getName().equals(cookiName)) {
        return cookie.getValue();
      }
    }
    return null;
  }

}
```

```java
@Configuration
public class WebConfig  extends WebMvcConfigurerAdapter{

    @Autowired
    UserArgumentResolver userArgumentResolver;

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {
        argumentResolvers.add(userArgumentResolver);
    }


}
```

之后写Controller时

```java
    public String toLogin(Model model,MiaoshaUser user
    ){
        model.addAttribute( s: "user",user);
        return "goods_list";

    }
```

参数类型直接是MiaoshaUser即可，代码更加简洁。