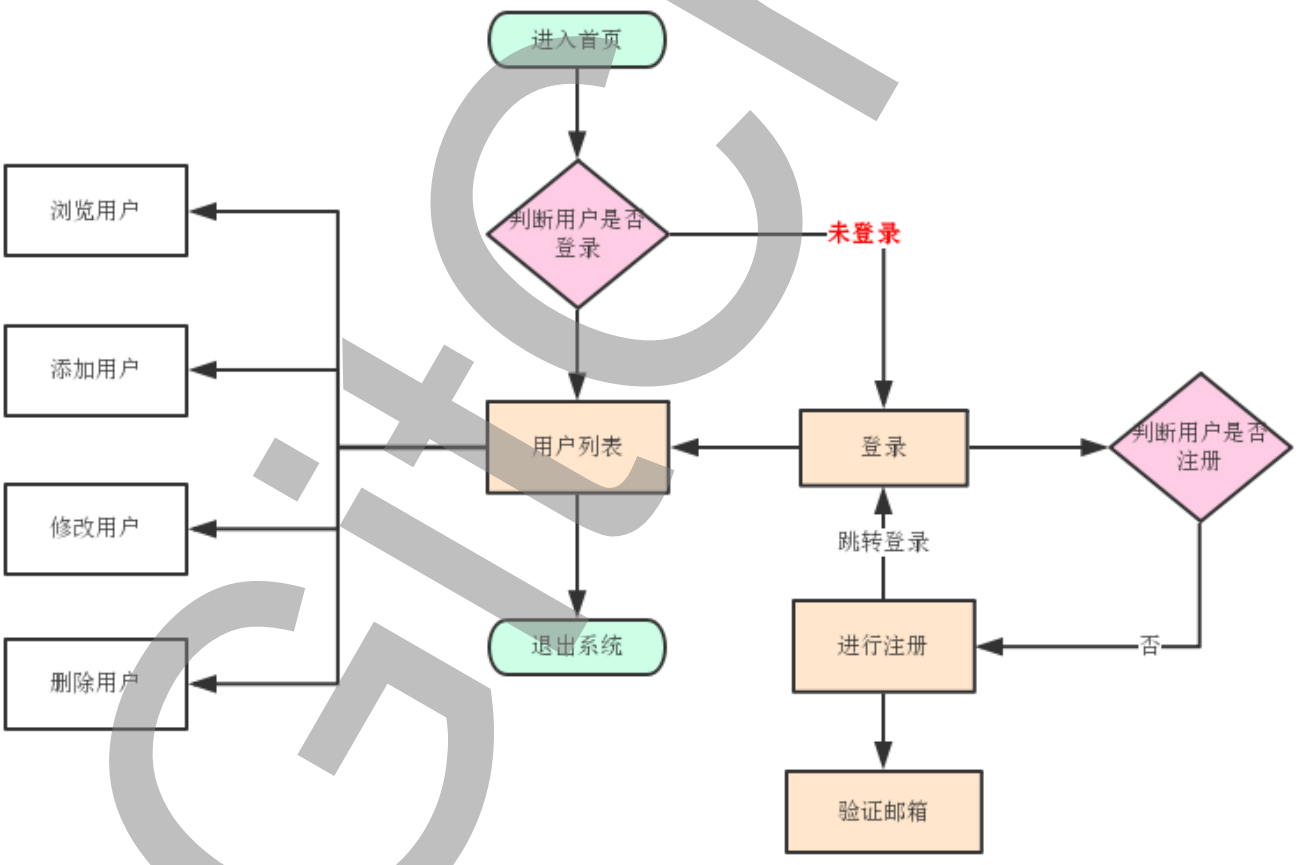
第 5-7 课：综合实战客户管理系统（一）

最后两课的内容是实践，结合前面课程的技术来做一个简单的用户管理系统，该系统包括以下功能：管理员注册、注册验证、管理员登录、管理员退出、添加用户、修改用户、删除用户、浏览用户信息等功能。

技术选型，使用 MongoDB 存储系统数据、使用 Filter 检查用户的登录状态、使用 Redis 管理用户 Session\ 数据缓存、使用 Spring Boot Mail 验证用户注册邮箱，使用 Hibernate-validator 做参数校验，使用 BootStrap 前端框架、Thymeleaf 模板，并且使用 Thymeleaf 进行页面布局。

功能设计

首先看一下用户管理系统的业务流程图：



- 访问首页，需判断用户是否登录
- 用户登录时判断是否注册，提示用户去注册
- 注册成功后，发送验证邮件
- 用户登录邮箱，点击链接验证邮箱
- 用户登录成功后，进入用户管理页面

- 用户管理页面可以对用户进行浏览，增、删、改、查等操作
- 用户可以单击“退出”按钮进行退出操作
- 每次的请求都会验证用户是否登录，如果 session 失效或者未登录会自动跳转到登录页面

从以上的内容可以看出用户管理系统的主要功能，如果在日常的工作中接到这样的一个需求，会怎么设计或者开发呢?

本节课程的开发步骤是：

- 开发数据库层的增、删、改功能
- 开发 Web 层代码，输出增、删、改、查的请求接口
- 页面布局、进行数据展示层的代码开发
- 结合上面前 3 步操作完成用户增、删、改、查功能
- 开发用户注册、登录、退出功能
- 注册成功发送验证邮件、单击邮件链接验证修改用户状态
- 进行 Session 管理，使用 Redis 管理用户的 Session 信息
- 添加自定义 Filter 对用户的请求进行验证
- 添加缓存、综合调试

前期准备

我们使用 MongoDB 做为数据库，需要有 MongoDB 数据库环境；使用了 Redis 管理 Session，需要有 Redis 环境；用户注册之后会发送邮件验证，需要准备邮件账户。

添加相关依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

从以上配置可以看出，我们使用 spring-boot-starter-web 组件处理 Web 层业务，使用 spring-boot-starter-thymeleaf 组件作为前端页面模板引擎，使用 spring-boot-starter-data-mongodb 组件操作 MongoDB 数据库，使用 spring-session-data-redis 组件和 Redis 交互，使用 spring-boot-starter-mail 组建处理邮件相关内容！

配置信息

数据库、Thymeleaf、Session 失效时间配置：

```
# mongodb 配置
spring.data.mongodb.uri=mongodb://localhost:27017/manage
# 测试环境取消 thymeleaf 缓存
spring.thymeleaf.cache=false
spring.session.store-type=redis
# 设置 session 失效时间
spring.session.timeout=3600
```

Redis 配置：

```
# Redis 服务器地址
spring.redis.host=192.168.0.xx
# Redis 服务器连接端口
spring.redis.port=6379
# Redis 服务器连接密码（默认为空）
spring.redis.password=
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=8
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=-1
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=8
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=0
# 连接超时时间（毫秒）
spring.redis.timeout=0
```

邮件配置：

```
spring.mail.host=smtp.126.com
spring.mail.username=youremail@126.com
spring.mail.password=yourpass
spring.mail.default-encoding=UTF-8
```

开发数据库层代码

```
public interface UserRepository  extends MongoRepository<User, String> {
    Page<User> findAll(Pageable pageable);
    Optional<User> findById(String id);
    User findByUserNameOrEmail(String userName, String email);
    User findByUserName(String userName);
    User findByEmail(String email);
    void deleteById(String id);
}
```

根据前期 MongoDB 的课程我们知道，集成 MongoRepository 会自动实现很多内置的数据库操作方法。

Web 层用户管理

Param 设计，在项目中创建了 param 包用来存放所有请求过程中的参数处理，如登录、注册、用户等，根据不容的请求来设计不同的请求对象。

分页展示用户列表信息

```
@RequestMapping("/list")
public String list(Model model,@RequestParam(value = "page", defaultValue = "0") I
nteger page,
                   @RequestParam(value = "size", defaultValue = "6") Integer size)
 {
    Sort sort = new Sort(Sort.Direction.DESC, "id");
    Pageable pageable = PageRequest.of(page, size, sort);
    Page<User> users=userRepository.findAll(pageable);
    model.addAttribute("users", users);
    logger.info("user list "+ users.getContent());
    return "user/list";
}
```

添加用户

```
public String add(@Valid UserParam userParam,BindingResult result, ModelMap model)
 {
    String errorMsg="";
    if(result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg=errorMsg + error.getCode() + "-" + error.getDefaultMessage()
+";";
        }
        model.addAttribute("errorMsg",errorMsg);
        return "user/userAdd";
    }
    User u= userRepository.findByUserNameOrEmail(userParam.getUserName(),userParam
.getEmail());
    if(u!=null){
        model.addAttribute("errorMsg","用户已存在!");
        return "user/userAdd";
    }
    User user=new User();
    BeanUtils.copyProperties(userParam,user);
    user.setRegTime(new Date());
    user.setUserType("user");
    userRepository.save(user);
    return "redirect:/list";
}
```

首先验证参数是否正确，再次查询此用户名或者邮箱是否已经添加过，校验无误后将用户信息保存到数据库中，页面跳转到用户列表页。

修改用户

```
public String edit(@Valid UserParam userParam, BindingResult result,ModelMap model
) {
    String errorMsg="";
    if(result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg=errorMsg + error.getCode() + "-" + error.getDefaultMessage()
+";";
        }
        model.addAttribute("errorMsg",errorMsg);
        model.addAttribute("user", userParam);
        return "user/userEdit";
    }

    User user=userRepository.findById(userParam.getId()).get();
    BeanUtils.copyProperties(userParam,user);
    user.setRegTime(new Date());
    userRepository.save(user);
    return "redirect:/list";
}
```

和添加用户的业务逻辑大体相同，最主要的是需要首先查询此用户信息，再根据前端内容进行修改。

删除用户

```
public String delete(String id) {
    userRepository.deleteById(id);
    return "redirect:/list";
}
```

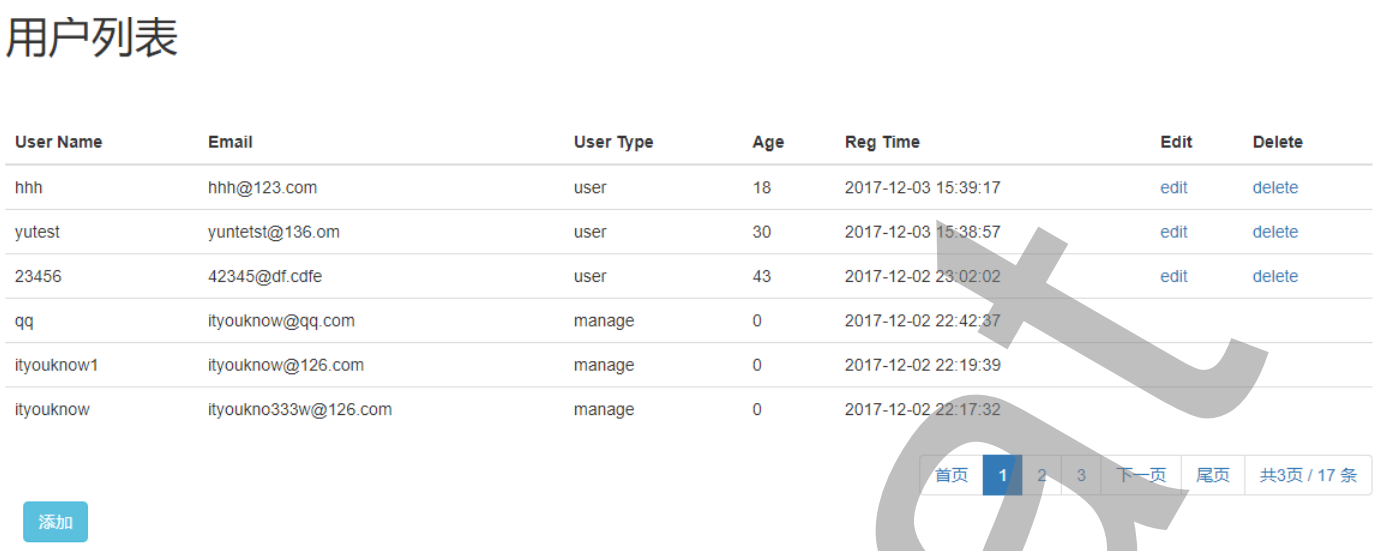
删除用户比较简单，直接调用 Repository.delete() 方法即可。

前端页面

用户列表

```
<h1>用户列表</h1>
<br/><br/>
<div class="with:80%">
    <table class="table table-hover">
        <thead>
        <tr>
            <th>User Name</th>
            <th>Email</th>
            <th>User Type</th>
            <th>Age</th>
            <th>Reg Time</th>
            <th>Edit</th>
            <th>Delete</th>
        </tr>
        </thead>
        <tbody>
        <tr  th:each="user : ${users}">
            <td th:text="${user.userName}">neo</td>
            <td th:text="${user.email}">neo@126.com</td>
            <td th:text="${user.userType}">User</td>
            <td th:text="${user.age}">6</td>
            <td th:text="${#dates.format(user.regTime, 'yyyy-MM-dd HH:mm:ss')}"></
td>
            <td><a th:href="@{/toEdit(id=${user.id})}"  th:if="${user.userType !='
manage'}" >edit</a></td>
            <td><a th:href="@{/delete(id=${user.id})}"  onclick="return confirm('确
认是否删除此用户? ')"  th:if="${user.userType !='manage'}" >delete</a></td>
        </tr>
        </tbody>
    </table>
    <div th:include="page :: pager" th:remove="tag"></div>
</div>
<div class="form-group">
    <div class="col-sm-2 control-label">
        <a href="/toAdd" th:href="@{/toAdd}" class="btn btn-info">添加</a>
    </div>
</div>
```

效果图如下:



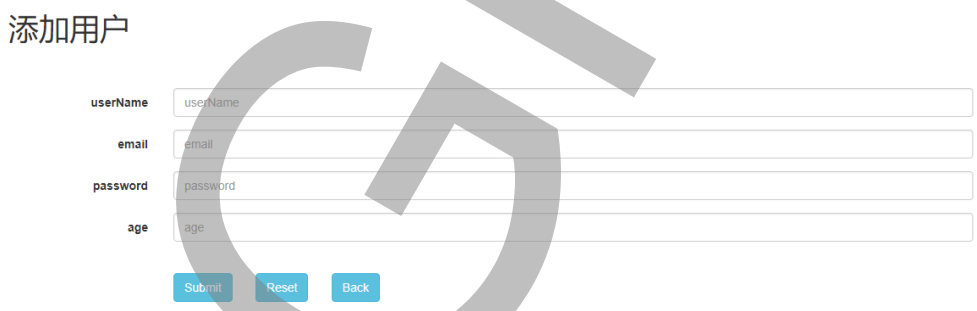
可以看出此页面的功能主要使用 Thymeleaf 语法循环遍历展示用户信息，并且给出修改和删除的链接；使用了 th:if 来根据用户的不同状态来选择是否显示编辑和删除链接；页面中有这么一句： ，其实就是引入了封装好的分页信息 page.html，前面课程已经介绍了分页信息这里不再多说。

添加用户

```
<form class="form-horizontal"   th:action="@{/add}"  method="post">
    <div class="form-group">
        <label for="userName" class="col-sm-2 control-label">userName</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="userName"  id="userName"
 placeholder="userName"/>
        </div>
    </div>
    <div class="form-group">
        <label for="email" class="col-sm-2 control-label" >email</label>
        <div class="col-sm-10">
            <input type="text" class="form-control" name="email" id="email" placeh
older="email"/>
        </div>
    </div>

...

    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <input type="submit" value="Submit" class="btn btn-info" />
                 
            <input type="reset" value="Reset" class="btn btn-info" />
                 
            <a href="/toAdd" th:href="@{/list}" class="btn btn-info">Back</a>
        </div>
    </div>
</form>
```

主要是输入用户相关信息，并提供提交、重置、返回的功能。

效果图如下:



修改页面和添加类似，大家可以参考示例代码。

注册、登录、退出

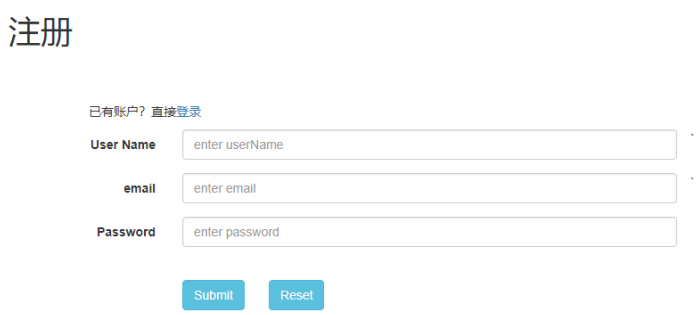
注册

注册的用户类型为 manage，后台添加的用户类型为 user，两者区分开来方便后续管理。

注册页面代码：

```
<div class="with:60%">
    <div style="margin: 10px 90px;">已有账户? 直接<a  th:href="@{/toLogin}">登录</a><
/div>

    <form class="form-horizontal"   th:action="@{/register}"  method="post" >
        <div class="form-group">`
            <label for="userName" class="col-sm-2 control-label">User Name</label>
            <div class="col-sm-6">
                <input type="text" class="form-control" name="userName"  id="userN
ame"  placeholder="enter userName"/>
            </div>
        </div>
        <div class="form-group">`
            <label for="email" class="col-sm-2 control-label">email</label>
            <div class="col-sm-6">
                <input type="text" class="form-control" name="email"  id="email" p
laceholder="enter email"/>
            </div>
        </div>
        <div class="form-group">
            <label for="password" class="col-sm-2 control-label" >Password</label>
            <div class="col-sm-6">
                <input type="password" class="form-control" name="password" id="pa
ssword" placeholder="enter password"/>
            </div>
        </div>

        <div class="form-group">
            <label   class="col-sm-2 control-label"></label>
            <div class="col-sm-6">
                <div th:if="${errorMsg != null}"  class="alert alert-danger" role=
"alert" th:text="${errorMsg}">

                </div>
            </div>
        </div>

        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-6">
                <input type="submit" value="Submit" class="btn btn-info" />
                     
                <input type="reset" value="Reset" class="btn btn-info" />

            </div>
        </div>
    </form>
</div>
```

效果图如下：



后端处理逻辑如下：

```
@RequestMapping("/register")
public String register(@Valid RegisterParam registerParam, BindingResult result, M
odelMap model) {
    logger.info("register param"+ registerParam.toString());
    String errorMsg = "";
    if (result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg = errorMsg + error.getCode() + "-" + error.getDefaultMessage(
) + ";";
        }
        model.addAttribute("errorMsg", errorMsg);
        return "register";
    }
    UserEntity u = userRepository.findByUserNameOrEmail(registerParam.getUserName(
), registerParam.getEmail());
    if (u != null) {
        model.addAttribute("errorMsg", "用户已存在!");
        return "register";
    }
    UserEntity user = new UserEntity();
    BeanUtils.copyProperties(registerParam, user);
    user.setRegTime(new Date());
    user.setUserType("manage");
    user.setState("unverified");
    userRepository.save(user);
    logger.info("register user "+ user.toString());
    return "login";
}
```

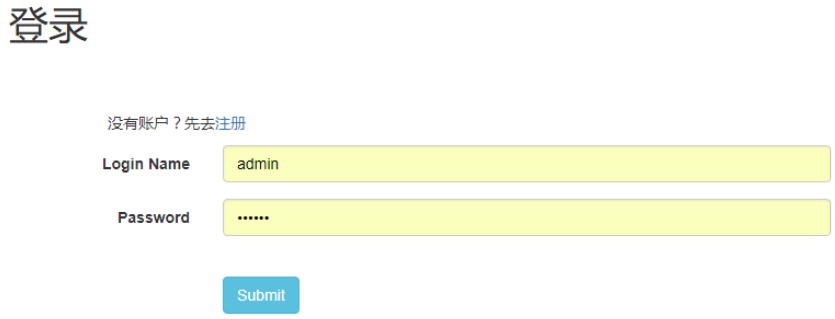
判断参数输入是否正确，用户是否已经存在，验证完毕后将用户信息存入数据库，并跳转到登录页面。

登录

登录页面代码：

```
<div class="with:60%">
    <div style="margin: 10px 90px;">没有账户？先去<a  th:href="@{/toRegister}">注册</a></div>

    <form class="form-horizontal"   th:action="@{/login}"  method="post" >
        <div class="form-group">`
            <label for="loginName" class="col-sm-2 control-label">Login Name</label>
            <div class="col-sm-6">
                <input type="text" class="form-control" name="loginName"  id="loginName" placeholder="enter userName or email"/>
            </div>
        </div>
        <div class="form-group">
            <label for="password" class="col-sm-2 control-label" >Password</label>
            <div class="col-sm-6">
                <input type="password" class="form-control" name="password" id="password" placeholder="enter password"/>
            </div>
        </div>

        <div class="form-group">
            <label   class="col-sm-2 control-label"></label>
            <div class="col-sm-6">
                <div th:if="${errorMsg != null}"  class="alert alert-danger" role="alert" th:text="${errorMsg}">

                </div>
            </div>
        </div>

        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-6">
                <input type="submit" value="Submit" class="btn btn-info" />
            </div>
        </div>
    </form>
</div>
```

效果图如下：



后端处理逻辑如下：

```
@RequestMapping("/login")
public String login(@Valid LoginParam loginParam, BindingResult result, ModelMap m
odel, HttpServletRequest request) {
    String errorMsg = "";
    if (result.hasErrors()) {
        List<ObjectError> list = result.getAllErrors();
        for (ObjectError error : list) {
            errorMsg = errorMsg + error.getCode() + "-" + error.getDefaultMessage(
) + ";";
        }
        model.addAttribute("errorMsg", errorMsg);
        return "login";
    }
    UserEntity user = userRepository.findByUserName(loginParam.getLoginName());
    if (user == null) {
        user = userRepository.findByEmail(loginParam.getLoginName());
    }
    if (user == null) {
        model.addAttribute("errorMsg", "用户名不存在!");
        return "login";
    } else if (!user.getPassword().equals(loginParam.getPassword())) {
        model.addAttribute("errorMsg", "密码错误! ");
        return "login";
    }

    request.getSession().setAttribute(WebConfiguration.LOGIN_KEY, user.getId());
    request.getSession().setAttribute(WebConfiguration.LOGIN_USER, user);
    return "redirect:/list";
}
```

首先根据用户名或者用户邮件去查找此用户，如果用户不存在返回并给出提示；如果找到用户判断用户密码是否正确，如正确，将用户信息存入 Session 中，并跳转到用户列表页。

退出

退出比较简单只需要清空 Session 中的用户信息即可：

```
@RequestMapping("/loginOut")
public String loginOut(HttpServletRequest request) {
    request.getSession().removeAttribute(WebConfiguration.LOGIN_KEY);
    request.getSession().removeAttribute(WebConfiguration.LOGIN_USER);
    return "login";
}
```

> 下一课我们继续介绍用户管理系统的设计研发。
>
> 点击这里下载源码。