**实现功能：首页、注册、登陆、登出、访问拦截验证**

**一、实现首页显示**

需求分析：用户访问首页"/","/index"，将最新的10条问题显示出来，主要包括两个方面内容：1问题发布者信息（头像，用户名），2问题本身信息（标题，具体内容，评论数，发布时间等）。点击发布者用户名进入该发布者首页，显示这个发布者的最新问题信息，形式和首页一样。

网站首页



发布者首页



实现概述：

Controller层：HomeController需要得到question对象list和每个对应的user对象。返回的是模板渲染的页面。

Service层(ServiceImpl层和其相对应)：UserService主要根据id获取user获取User对象、QuestionService主要获取最新的10条问题List<Question>

Mapper层：UserMapper根据id查User对象，QuestionMapper根据userId和分页情况返回最新的几条对象List<Question>。

Model层：User和Question对象

具体实施：

1、数据库设计

数据库包目前需要用的有三个表：user、question、login\_ticket。

1.1 user表初初始化

DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(64) NOT NULL DEFAULT '',

`password` varchar(128) NOT NULL DEFAULT '',

`salt` varchar(32) NOT NULL DEFAULT '',

`head\_url` varchar(256) NOT NULL DEFAULT '',

PRIMARY KEY (`id`),

UNIQUE KEY `name` (`name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

其中包括主键（id），用户名（name），密码(password)，密码盐(salt)、头像地址（head\_url）。

1.2 question表初初始化

DROP TABLE IF EXISTS `question`;

CREATE TABLE `question` (

`id` INT NOT NULL AUTO\_INCREMENT,

`title` VARCHAR(255) NOT NULL,

`content` TEXT NULL,

`user\_id` INT NOT NULL,

`created\_date` DATETIME NOT NULL,

`comment\_count` INT NOT NULL,

PRIMARY KEY (`id`),

INDEX `date\_index` (`created\_date` ASC));

其中包括主键（id），问题标题（title），问题内容(content)，问题发布者id(user\_id)、发布日期（created\_date）。评论数（comment\_count,这个其实是冗余设计，可以直接查comment表取到个数也可以，但是效率相对低了）

1.3 login\_ticket表初初始化

DROP TABLE IF EXISTS `login\_ticket`;

CREATE TABLE `login\_ticket` (

`id` INT NOT NULL AUTO\_INCREMENT,

`user\_id` INT NOT NULL,

`ticket` VARCHAR(45) NOT NULL,

`expired` DATETIME NOT NULL,

`status` INT NULL DEFAULT 0,

PRIMARY KEY (`id`),

UNIQUE INDEX `ticket\_UNIQUE` (`ticket` ASC)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

其中包括主键（id），秘钥拥有者id（user\_id），秘钥内容（ticket，这个其实就是一个随机字符串），过期日期(expired，这个时间用于给定有效期)，秘钥状态(status，为0位有效，其他就失效)

数据库的初始化可以直接在navicate中进行，也可以用测试类（前提是在mysql已经建立好了数据库，并和配置文件db.properties对应好）在InitData.java中，将sql文件放到test的resources中，[使用注解@Sql("/init-schema.sql")](mailto:使用注解@Sql(%22/init-schema.sql%22))，这样在测试类运行的时候会执行sql文件内容，建立表格。

2、采用MVC分层思想开发，基本的一般包括：

Mapper:数据库交互层，目前一般使用接口加注解方案，如果sql语句比较复杂就和xml进行绑定写动态sql语句等。

ServiceImpl、Service：业务层主要负责整个项目的逻辑处理代码，属于核心部分，调用mapper层和数据库交互，被controller层调用返回处理结果。

Controller：负责接收前端传来的请求，调用service层进行处理，进行相应的跳转

Modle:用于存放实体类，一般和数据库中的表对应创建对象。

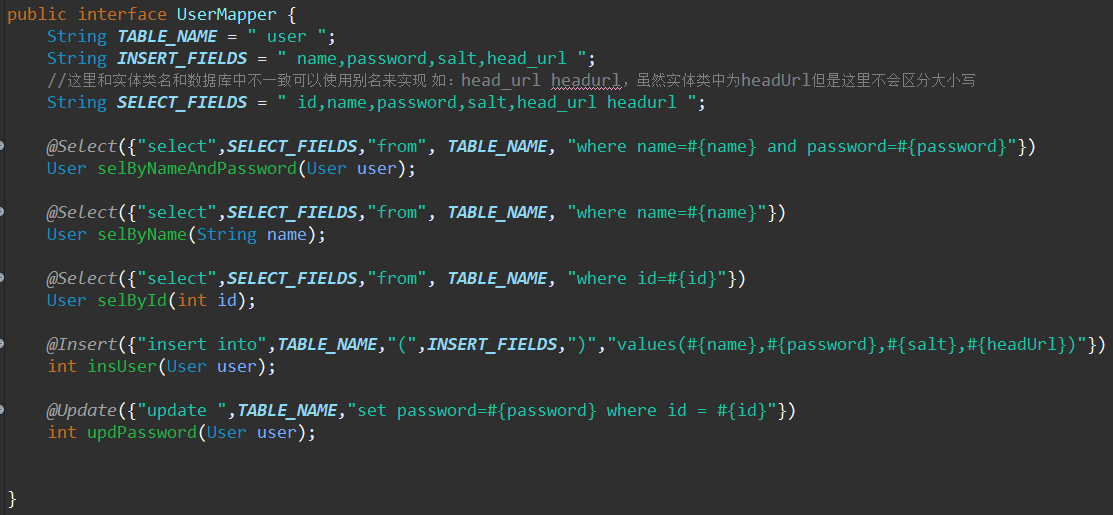
流程一般是写model mapper service serviceImpl controller

2.1 对数据库表user实现增删改查

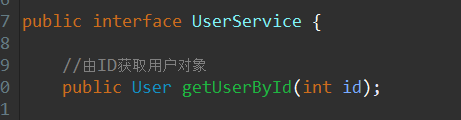
先在model层创建实体类user,属性和user表字段匹配，然后再mapper层创建接口UserMapper，通过注解写sql语句,如下图。Mapper包会被扫描器扫描形成和接口名称对应bean-🡪userMapper(首字母变成小写)，这样在其他地方就可以直接注入就行。

注意点：首先需要注意的是可以定义常用的变量如表明，插入字段，查找字段等，跟sql语句解耦，这样修改的时候就很方便了。其次是在SELECT\_FIELDS中因为实体类和数据库命名的差别如head\_url 和 headUrl,这里应该使用别名来做如：

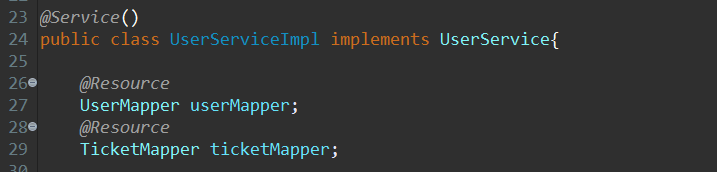
head\_url hearurl(虽然实体类是headUrl,但是这里可以不区分大小写)。这个问题也可以在配置文件中配置解决（目前没有配置，因此使用别名的方式）



2.2 service定义接口、serviceImpl定义具体定义实现类。下面例子为UserService接口（和普通接口一样写就行）。



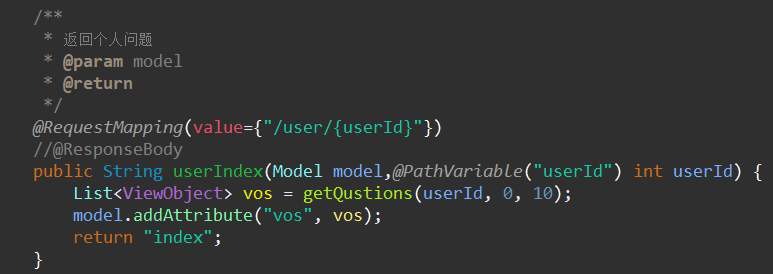
UserServiceImpl类实现上面接口并重写其定义的方法getUserById,这个需要从数据库中读取。这里通过@Resource可以直接将刚才定义的userMapper注入，然后值机调用函数就行了。还要注意的一点是类上面要加@Service（后面可以带上括号写上作为javabean的别名，否则默认为类名首字母变小写）使得自己能作为bean被其他层注入使用（如controller层），同时因为这一层使用到了注解，因此在spring配置文件中注解扫描包要有com.zju.service这个包。以后那个文件用到了spring的注解也要这么做。



2.3 controller中新建HomeController类，同样类上面@Controller（这里是要被springMVC扫描），调用UserService就直接通过注解依赖注入。注意点有两个：1、@Resource后面参数name为指定UserService接口要实现的类名（bean中），这样如果要修改实现类只需要修改name的参数就行。2、在controller中不能写成UserServiceImpl userServiceImpl然后上面加@Resource注入，这里类型只能为接口（原因不明，但是会报错，和代理有关）

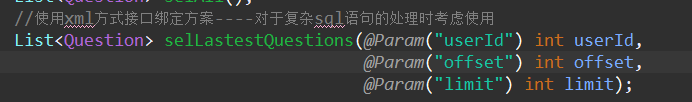


controller中方法：@RequestMapping后面跟着访问地址，返回值类型为String的时候，如果加了@ResponseBody就返回字符串“index”给前端，如果不加，根据模板引擎的配置，会去访问index.html,同时可以通过modle携带参数过去，在html中就可以通过把参数通过velocity语法加入到页面中，最后整个html文件传给前端。

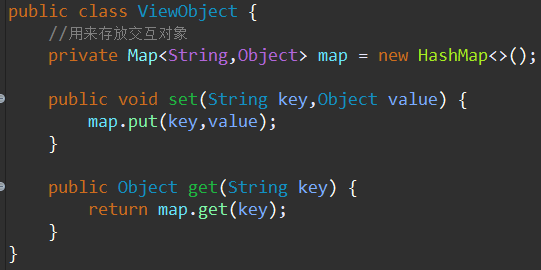


3、首页显示的实现

3.1 首先和如上和user一样编写question相关的model/mapper/service等，然后通过测试类中的InitData.java初始化建立数据库的同时，在其方法中调用userMapper和questionMapper插入几条初始数据。（需要注意的是question的查找最先10条消息的函数，如下所示，sql语句复杂用到了xml和接口绑定的方案，@Param用于传递多参数情况）。



3.2 在model层建立ViewObject类用来包装多个不同数据到view层（就是模板渲染的层，html文件）---如首先的每一条问题的显示都需两个内容问题类Question（只有发布者id,没有发布者姓名头像等其他信息，要根据id从数据库查出来，再捆绑发给前端） 和发布者类User,就可以统一装入一个ViewObject中进行封装捆绑，方便传递。



3.3 首页要显示的内容就是从数据库查出最新的10条问题，并且把其发布者信息一起显示。首先插入10条问题详情（question对象），由list接收，然后遍历list中每个Question，根据其内userId查找对应的User对象，然后两个对象封装成一个ViewObject,最终得到10个ViewObject放到list中返回给View层。View中通透velocity读出来并将内容填入html对应的位置中（前端的工作），让后整体返回给浏览器。

**二、实现注册、登陆、免登陆、登出、拦截验证等**

需求分析：用户访问“reglogin”（前端来说是点击注册按钮，下同）进入登录页面，用户访问"/reg"进行注册，用户访问“/login”进行登录，用户访问“/logout”退出登录，注册和登陆成功重定向到首页（并在主页中显示当前登录用户的头像---在拦截其中实现），否则返回注册登陆页面并提示错误信息，点击退出登录后也返回到首页。



实现概述：

Controller层：LoginController得到用户名和密码，交给Service层的操作，并根据返回信息判断是否成功然后进行选择跳转，如果成功进行cookie插入和设置。

Service层(ServiceImpl层和其相对应)：UserService在注册和登陆时候对信息进行校验（主要通过用户名查找数据库中的用户来判断），如果不合法向上层抛出不合法信息，如果成功就新建秘钥对象并抛出秘钥给上层。

Mapper层：UserMapper主要是新插入用户对象，根据用户名查对象，ticketMapper主要是插入新的ticket，根据ticket内容获取到数据库中ticket对象（用于拦截器判断ticket的有效性），将ticket设置成失效（退出登录的时候）

Model层：User对象、Ticket对象、HostHolder里面存放线程容器，拦截器中存放登陆用户的对象信息。

Intercepter层：第一层拦截所有（目的是为了获取登陆用户的ticket），首先查找是否携带标签为ticket的cookie，有的话就利用ticketMapper层将这个对象根据ticket找出来，然后判断其有效性，有效的话根据userMapper将这个用户对象获取到，然后放入modelAndview中，进行首页模板渲染的时候直接使用。第二层拦截为对页面权限设置，当访个人主页的时候要求先登录才能使用，未登录用户直接跳转到登陆界面。

1、用户注册

主要在service层（包括了接口层和实现类层，以后都是）。Controller接收前端传来的参数username 和password传给service层中的regist函数。

首先进行非空验证（虽然前端表单校验已经验证过了，但是有可能写脚本来注册，因此要双重验证），有问题就返回问题（通过map返回），正常就由用户名去查有没有，如果返回有说明用户名已经被注册，返回错误信息。都可以就新建对象，输入用户名，随机分配头像地址，并对密码进行加密处理。

密码不能明文保存在数据库中，这样如果数据库泄露，密码就泄露了。这里采用密码加上一串随机字符串再MD5加密方式（直接MD5加密也有问题，因为人们常设置的一些密码都被解密过了，反破解一下就出来了，如果加上随机数字符串就可以避免）。

过程如下：先UUID生成随机字符串取5位，给User中的salt属性，然后password+salt进行MD5加密，得到的字符串赋值给User中的password。最后将user差如数据库保存，注册成功。Controller层重定向用户到主页。

2、用户登录

和注册一样也要先进行一系列验证，然后到密码验证。首先从从数据库根据用户名取出User,的到salt和前端的传来的password相加并MD5加密，然后和数据库总的User中的password比较是否一致。

3、用户访问免登陆、登出

这里要用到数据库中的login\_ticket表。以用户登录为例子，如果一个用户进行账户密码登录，验证成功后就新建一个Ticket对象，其中属性userId存入用户id,属性ticket存入随机生成一个字符串作，staus默认有效为0，expired为到期时间，设置为由当前时间往后一段时间（这里设置为3天），然后存入数据库中，并把ticket这个随机字符串抛给controller层。在controller层中创建cookie,存入这个信息，同时如果用户点击记住密码还应设置cookie有效期，默认为关闭浏览器就失效。

用户再次访问会携带ticket这个cookie,系统就可以验证其有效性让其免登陆，在第4点访问拦截中说明。

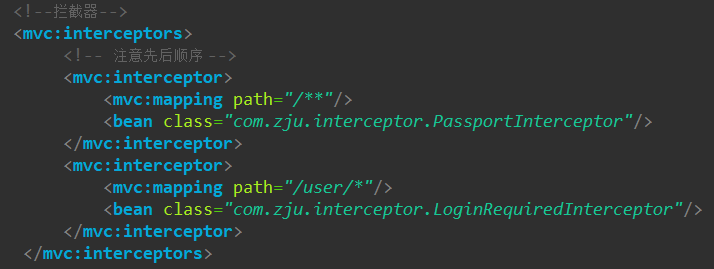
用户如果在登陆状态要登出，就先获取cookie中ticket然后通过ticket查找Ticket对象，将其status改为1，然后重定向回主页。

4、访问拦截验证

拦截器在在整个访问过程中有三个切入点：①整个访问的最开始②从controller到View层③整个访问的最后。

目前通过拦截实现两个功能：1对于每个访问都拦截一下判断其是否携带有效ticket,如果是则认为是登陆用户（在主页显示器个人头像）2对某些网页进行权限设置，这里实现用户详情网页只能登陆状态才能访问，非登陆用户访问会跳转到登陆页面去登陆。

4.1创建拦截器的包和类（注解扫描可以不加，虽然里面用到了注解），然后在springmvc.xml配置文件中配置拦截器，注意这里配置的先后顺序影响了拦截顺序。

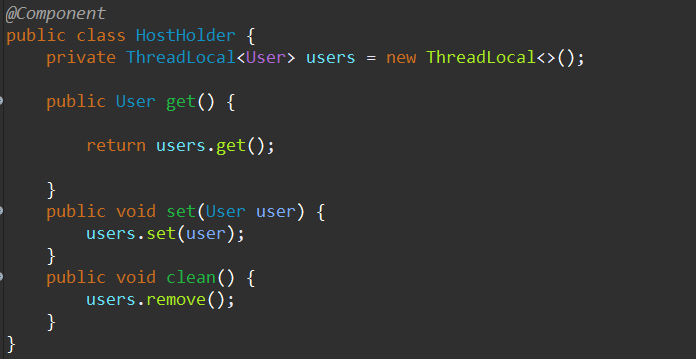


4.2 对于用户登录状态拦截验证（第一个）

这里配置成拦截所有,这里注意的是用的是/\*\*可以拦截所有访问，如果是/\*不能拦截带二级目录的访问如/user/11这样的。这一访问指示为了得到用户的ticket，不做任何拦截处理最后都会放行。

在preHandlec处（上述第一个切入点），查看用户是否有cookie中是不是有有效的ticket，有的话通过Ticket的userId可以找到查到User对象。

（重要）在model中创建HostHolder类，通过线程容器可以在整个线程内进行共享变量，同时将这个类通过@Component（用到了spring注解,需要在配置文件中假如注解扫描）放到bean中，那么整个框架中都可以进行依赖注入获得。线程间还能各不干扰。



将在第一个拦截器中获得的user放入HostHolder中，整个线程就能轻易获得登陆用户的信息。因此如果是登陆用户的话HostHolder中的内容就不为空，非登陆用户就为空。

在postHandle（第二个切点和View交互）处就可以直接从获取User对象，通过ModelAndView传到View层，View层就能获得用户头像，然后显示说明已经是登陆用户。

在afterCompletion（整个访问结束时—返回html之前）需要将这个HostHolder的内容清空，因为已经用完了，访问即将结束。

4.2 网页访问权限设置（第二层）

根据前面的配置可知只有访问特定页面如/user/\*页面时候才会被这个拦截，判断是不是登陆用户，这里的判断非常简单，先注入hostHolder，然后从里面拿看下有没有User用户，有的话就是登陆用户，直接放行。如果为null就是非登陆状态，重定向回登陆页面。这里多了一个小功能：先记录下当前要访问的URI，通过参数形式传给登陆页面，登陆页面记录下来，当这个用户成功登陆后直接重定向回刚才被拦截的页面（我们在之前设置的登录成功是直接重定向到主页）。