

上面的一些难懂地方的解释：

1. **将验证码存入缓存**，在检查时**不用查询本地session**，**直接查询缓存**，和缓存中的数据能对应，就通过；
2. 在网页头部每次**请求**都**会查询用户的登录凭证**，已显示用户登录后的网页头部，或没登录的头部，访问频率非常高；处理每次请求, 只要是用到的用户都会存入缓存中，可能是方便点击用户头像来得到用户页面信息

(但我觉得只用将自己登陆的用户存入缓存中就好了，因为可能更改头像等自己的用户信息，以及很多其他对自身的操作，你改不了别人的，将其他用到的用户放入缓存就有点多此一举了，因为其他用户的访问没那么频繁，存入缓存也需要访问数据库，从数据库中取，还要写入缓存，更加耗时（个人理解，那要是这样就不用对user类更改操作方法等，只需在hostLocal中封装一下，每次调用先查询，没有就写入缓存）

具体操作：

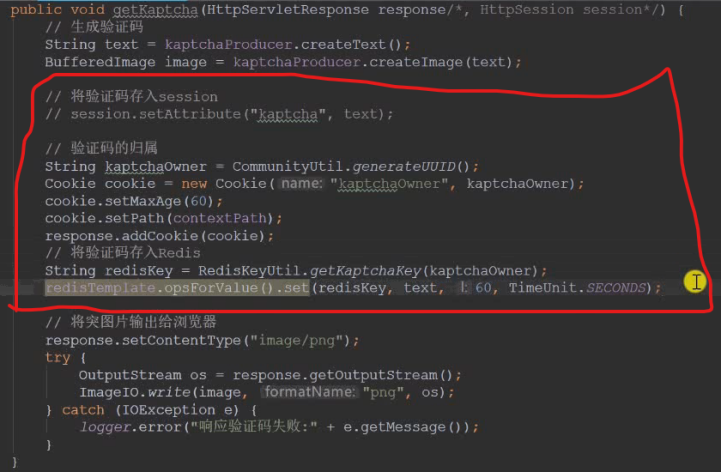
验证码：

先**设置redis中的key**，验证码将各个用户的区别开，但由于还未登陆，不能用用户的id区分，可以设置一个owner字符串，在获取验证码时，**随机生成一个owner字符串**，将其通过cookie传入给用户浏览器端，当你点击登录需要验证验证码是否正确时，将cookie传给服务端



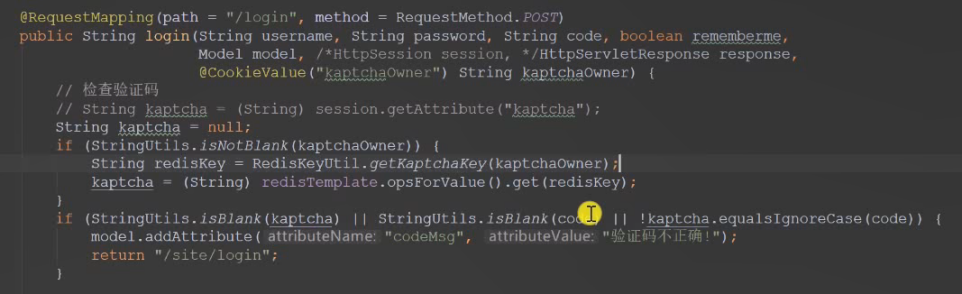
Controller层，在获取验证码时的更改：

通过uuid工具类生成随机字符串KaptchaOwner，传给cookie，设置cookie的有效范围为项目内有效，60s，将验证码传入redis中



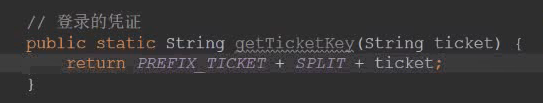
检查验证码操作：

首先**判断**当前**验证码kaptachaOwner的key对应的value是否为空**，因为你的key是通过他来算的，他又是通过cookie来传入，若时间过期就直接返回验证码不正确（验证码的时间限制），之后在判断验证码是否正确



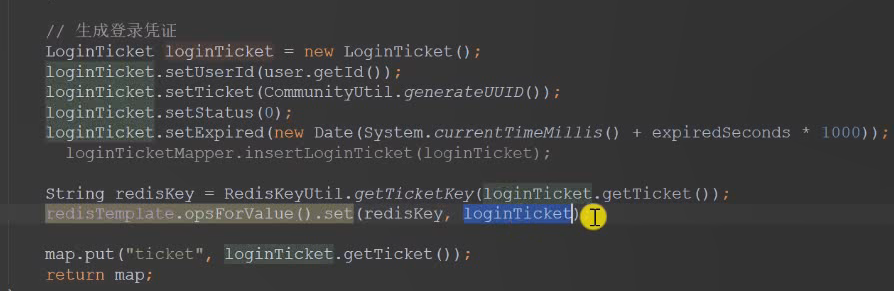
**使用redis存储登录凭证(**ticket)（登录凭证不删除，因为后期可能统计，比如一年你登录了多少天等）：

设置key对应的名称：

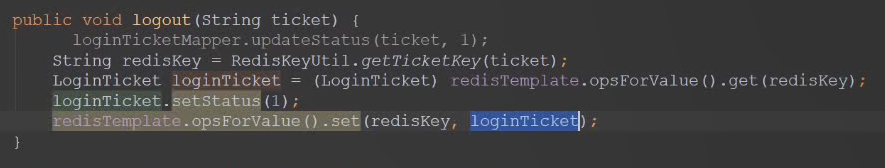


UserService层生成登录凭证：

实体类不用改，只是在加入的时候，不传数据库，传给redis，也是对应的string类型的，因为redis会将其(ticket)自动转化为json格式



比如在更改登录状态时，取出key对应的value可以强转



**缓存用户数据**

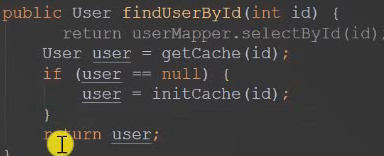
也是string格式，同上，在获取时可以强转为对应的类型（删除删的是key）

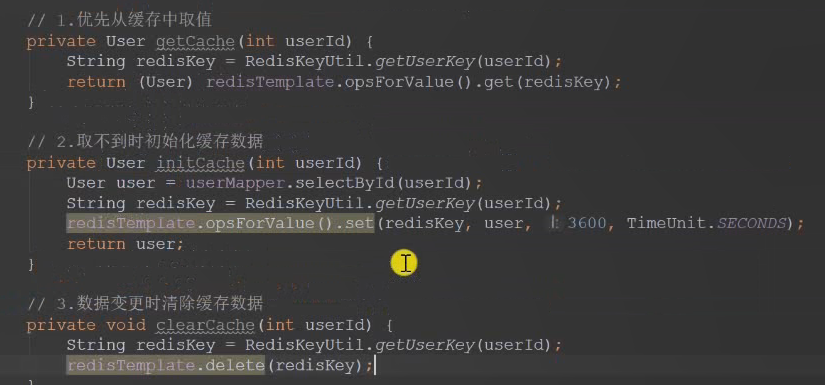
数据变更时先删除缓存，在把缓存初始化

比如**查询**方法：

**先从缓存中取，没有再初始化**

**更新的时候只需删除缓存(更新的话会有并发问题)即可，在查询的时候才可能会写入缓存（对应的key未找到）**





如下图，先更改，再删除

