第四次作业：

结合SSM框架和前端界面的嗖嗖移动服务大厅

1. **框架的搭建**
   1. **目录结构**

基于网上和课堂上的优秀分层范例，我将原来的嗖嗖移动服务大厅分成了一下几个层次。（如图1-1）

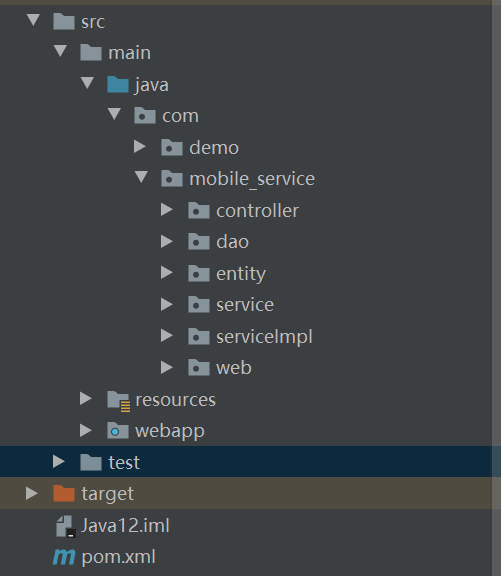


图1-1

首先，在建立maven的webapp项目的基础上，我将项目分成了main和test两个的文件夹，其中test主要是存放的是利用junit进行单元测试的测试类；main存放的是所有的项目代码。main文件夹又分成了java、resources、webapp三个文件夹，其中java文件夹设置成root sources文件夹，resources设置为root resources文件夹，所有的配置文件都存放于此，webapp文件夹存放的是前端页面设计的jsp文件以及针对网页的配置文件。

而java.com.mobile\_service的包中分为了controller、dao、entity、service、serviceImpl和web六个文件夹，分别存放的是应用spring mvc的控制器类，定义数据库操作的接口、业务逻辑所需的实体类、业务逻辑相关代码所在类、连接数据库具体操作和业务逻辑的实现类以及实现servlet的类（web文件夹主要存放未用框架前实现jsp逻辑跳转、信息提交等功能的类）。

* 1. **目录分层的好处**

通过不断地将类细分，归入不同的包中，有利于整理代码思路，提升封装性，减少代码冗余，以便更好地进行后期维护。

* 1. **环境的配置**

SSM框架所需要的jar包都是通过Maven管理，在pom.xml文件中添加配置语句进行相应的加载。配置文件中有清晰的注释写明jar包的使用情况，如有必要可以查看源代码。

1. **Mybatis的应用**

**2.1 接口的设计**

在第三次作业的接口设计基础上，我按照用户和后台数据维护的分类设计了两个接口。图2-1是涉及用户的数据库操作接口，第一部分针对的是userInfo表，记录基础用户信息，第二部分针对的是充值表格、消费表格以及对零散的某几列的服务进行修改的接口，第三部分是关于packageUse表格进行的接口设计。受助教的启发，针对userInfo表和packageUse表设计了外键，以保证数据增删的统一性，这样的设计一定程度上减少了接口的设计数量和降低了mysql的映射文件的书写难度。

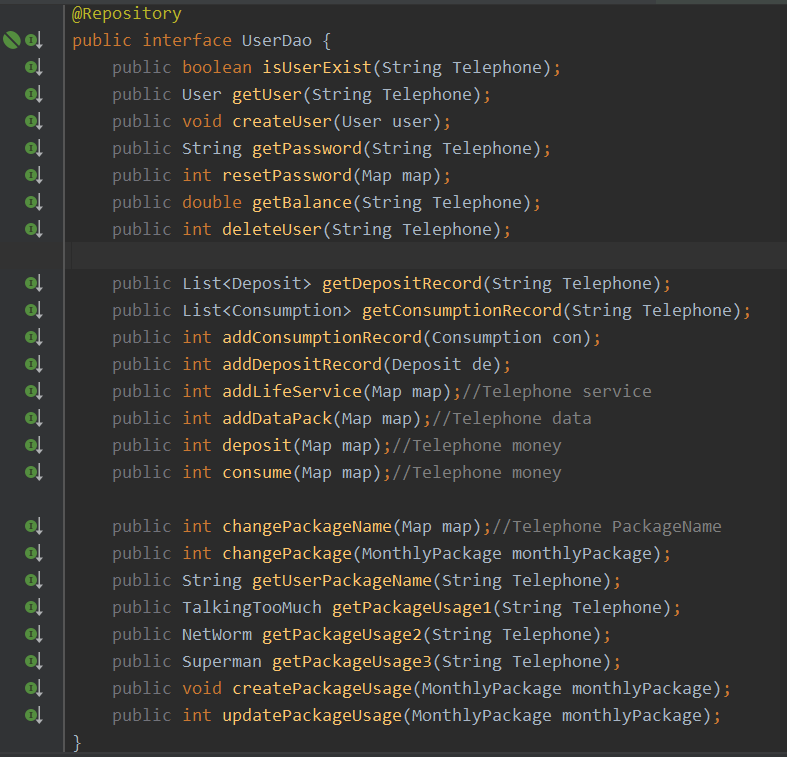


图2-1

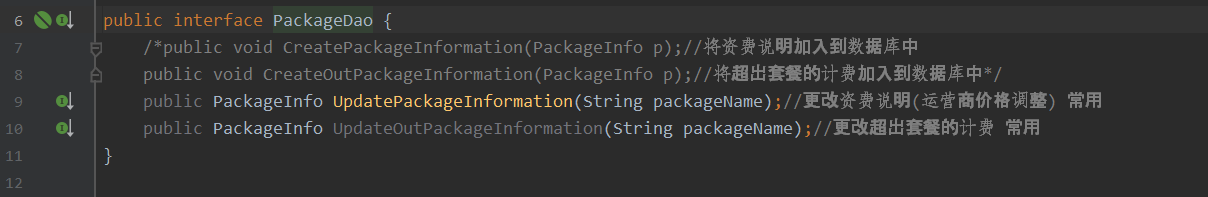


图2-2

**2.2 spring+mybatis的配置**

首先我们需要通过配置文件，将数据库的基础信息和环境进行配置。如图2-3。

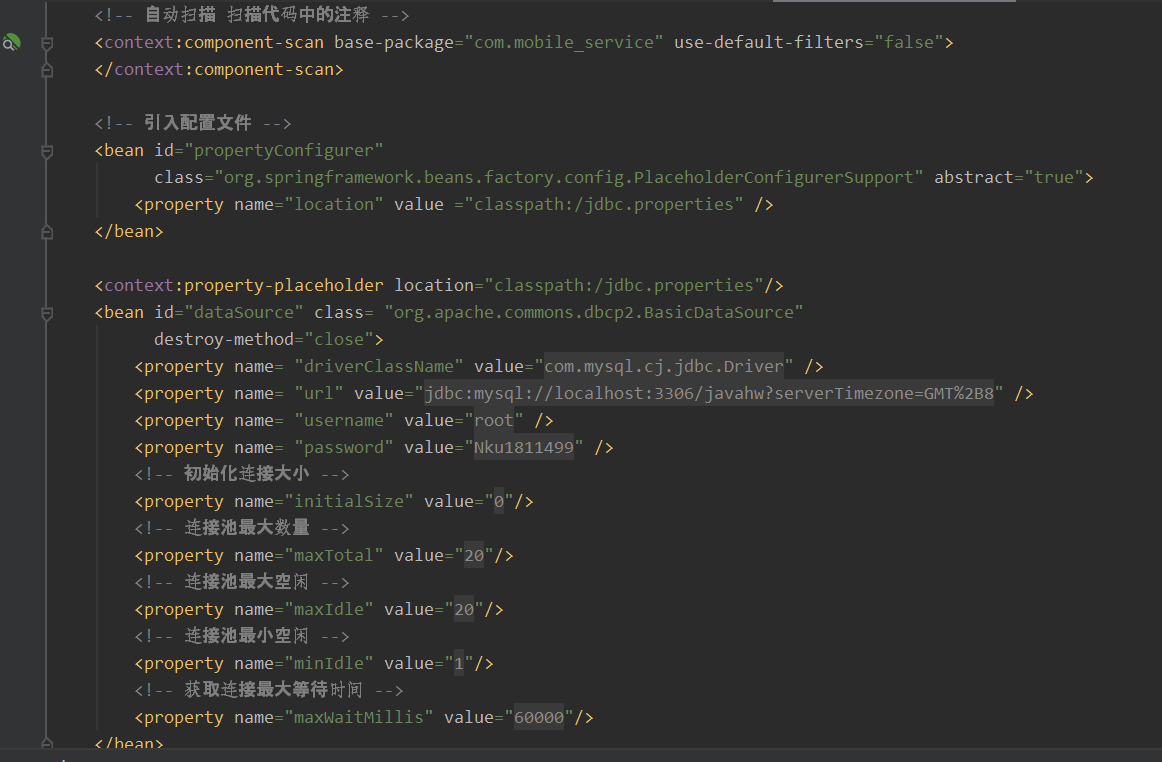


图2-3

其次，我们需要将spring和mybatis结合起来。在没有spring之前，我们需要通过mapper的映射，将设计的数据库操作接口在配置文件中进行映射，然后还需要实现工具类，以完成数据库的连接，每次操作数据库时建立数据库的会话工厂（sessionFactory）等功能，比较繁琐。但是整合了spring之后，我们就可以将这个工具类变成配置，让spring自己去找dao接口所在的包名，对应的sql语句的动态解析mapper文件，并设置事务管理器和sqlSessionTemplate等。如图2-4。当然，还可以在配置文件中实现通过注解完成事务处理的配置。如图2-5。



图2-4



图2-5

另外，我们还需要把接口，MySQL的mapper解析文件，以及接口的具体实现类和业务逻辑类建立联系。如图2-6。这里配置了四个类，分别对应的是UserDao的直接数据库操作实现UserServiceImpl类和间接数据库操作的业务逻辑UserService类，以及packageDao的两个实现类PackageServiceImpl、PackageService。

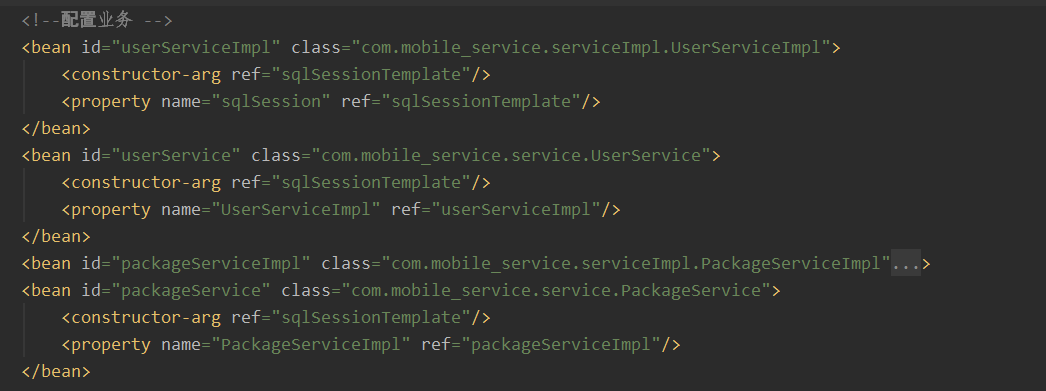


图2-6

**2.3直接数据库操作类的实现**

和之前的数据库操作不同，我的MySQL语句都放在UserDao.xml文件当中，而UserDao又不可以被实例化，所以我们需要UserServiceImpl这样也给类去逐一地对UserDao进行实现。如图2-7。首先我们需要声明UserDao和SqlSessionTemplate，这才可以使UserDao接口中每一个函数的具体操作被sql的session获取到并执行增删改查操作。以getUser这个函数为例，我们只需要通过SQLSessionTemplate获取到UserDao对应的mapper后，就可以执行对应的函数。

这个过程中看似是接口实例化去调用具体的函数，但实际上是spring和mybatis在背后帮助我们完成了接口的实例化，MySQL语言的注入、解析、执行。

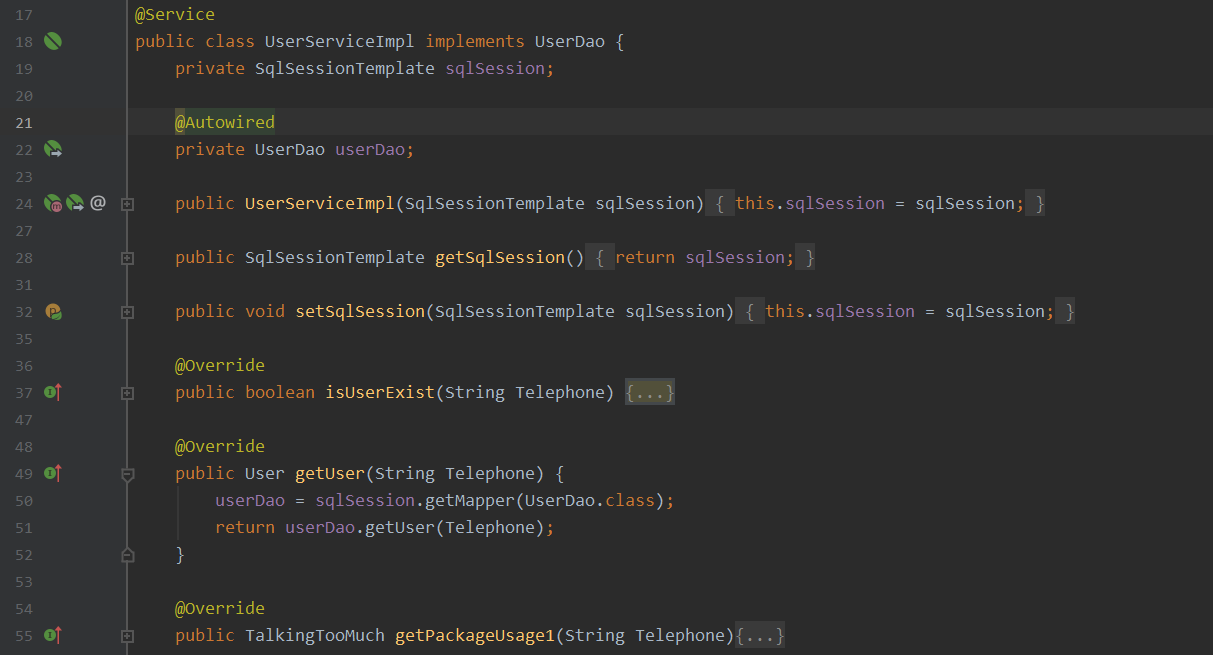


图2-7

**2.4 业务逻辑的实现（间接操作数据库）**

但在项目的具体使用上，我们可能需要同时进行多张表的增删改查，又或者我们要实现更高级的、有目的性的增删改查。因此我还设计了一个UserService类。原来承载了整个嗖嗖服务大厅的ServiceHall类的很多代码都被封装成了UserService类中的函数。以新注册一名用户为例。如图2-8。可以看到，当我们得到一个新用户的时候，我们需要首先判断，它是不是真的是新用户，因此以电话号码为主键去获取数据库中的用户，如果结果为空则说明该用户的确是新用户，这是判断条件，也是第一次用到直接数据库操作。

然后我们需要创建新用户的信息记录，即在userInfo表格中插入一条记录，即createUser函数所做的事情，第二次进行直接数据库操作。除此之外我们还需要将该用户的套餐信息存入packageUse表中，createPackageUsage函数做第三次数据库操作。因为在注册的时候，用户需要充值，那么充值记录表也需要增加信息，第四次进行数据库操作。

由此可见，一个简单的增加新用户就需要进行四次不同的数据库操作。如果将他们封装到一个函数里，就可以方便后续的信息展示和控制器调用。

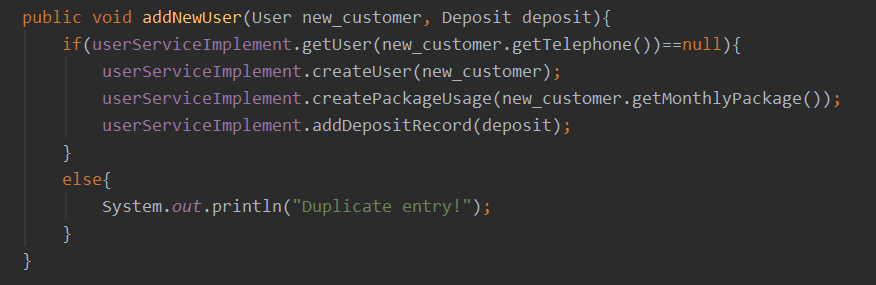
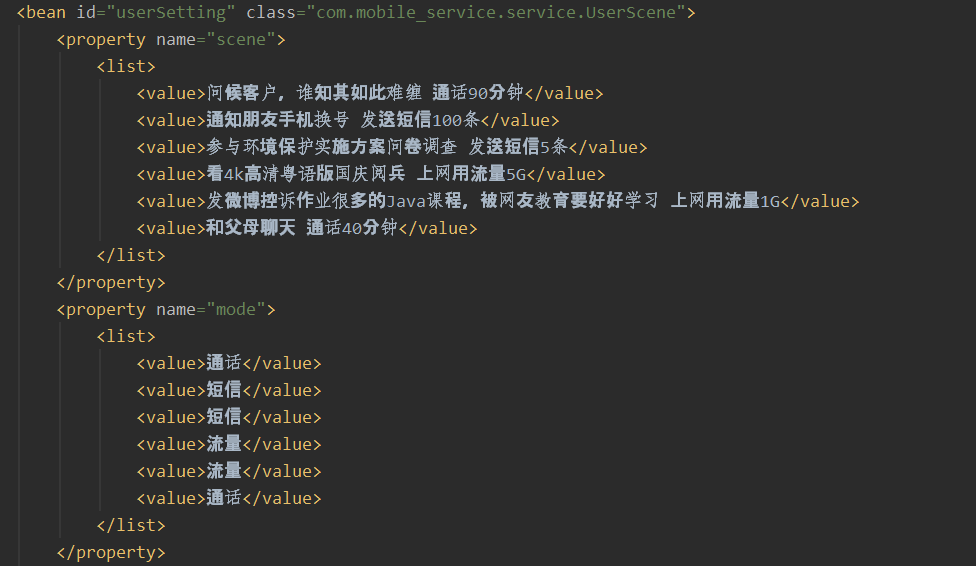


图2-8

1. **Spring的应用**

Spring框架在此次作业中更多的是起着连接的作用，以及在细节处实现优化。但由于时间关系，我没能完全学会用Spring框架进行更好地代码异常维护、事务处理。值得一提的是我利用Spring的默认构造和属性注入方法，帮助我实现了对实体类信息的赋值，从而减少了简单机械的赋值代码的编写。如图3-1。



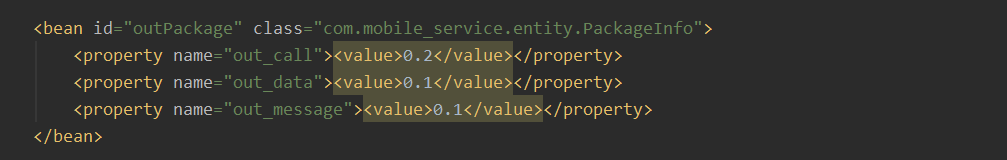


图3-1

但在实现Spring的属性注入时，我发现如果遇到想基类和子类分别进行注入的情况的话，这时候用属性注入无法达到目标。我们所认为的是如果我对基类的数据成员进行了属性赋值，那么子类也可以随之获得该参数。但实际上Spring的属性注入方法是针对类的实例化对象而言的，基类被注入属性后的对象和子类被注入属性后的对象不是同一个，因此在进行相互赋值的时候就会出现错误。但由于我的项目中套餐的类设计正是需要利用虚基类的多态性，因此这时候的属性注入方法无法实现我的目标。并且，更糟糕的是，我发现必须要调用Spring的配置文件中对应的bean才可能完成这种属性的注入，否则是没有效果的。几经周折，我采取了以下办法：先通过一个通用类的属性注入获取我需要得到的默认属性值，然后再在这些有继承关系的类的构造函数中调用这个通用类的bean，再进行赋值。

尽管我觉得这不是一个十分理想的方法，但介于目前的学识水平，这个办法至少解决了我项目中信息无法及时同步的难题。

1. **Spring MVC的应用**

**4.1 Controller类的编写**

为了书写方便，我是通过注解的方式完成函数与对应网址的映射。先介绍一下我的controller类的设计。

首先我设置了一个总的界面homepage，在该界面上用户可以选择前往注册界面、登录移动服务大厅界面、使用服务界面。而RequestMapping中的“/homepageRedirect”就是处理这个跳转的函数。如果选择注册的话，那么“/loginPage”负责处理所有注册需要完成的信息交换，页面跳转等。其中注册页面会有一个页面跳转查看资费说明，由“/packageInfoDisplay”处理。如果选择登录的话，首先会有“/loginCheck”这个函数帮忙处理能否登录，忘记密码等问题，成功登录后的移动服务大厅的各项业务处理交给“/Service”去处理。如果是想使用服务的话，同样会先有“/useCheck”完成使用服务资格的检验，然后交由“/UserScene”处理使用服务中的各种操作。

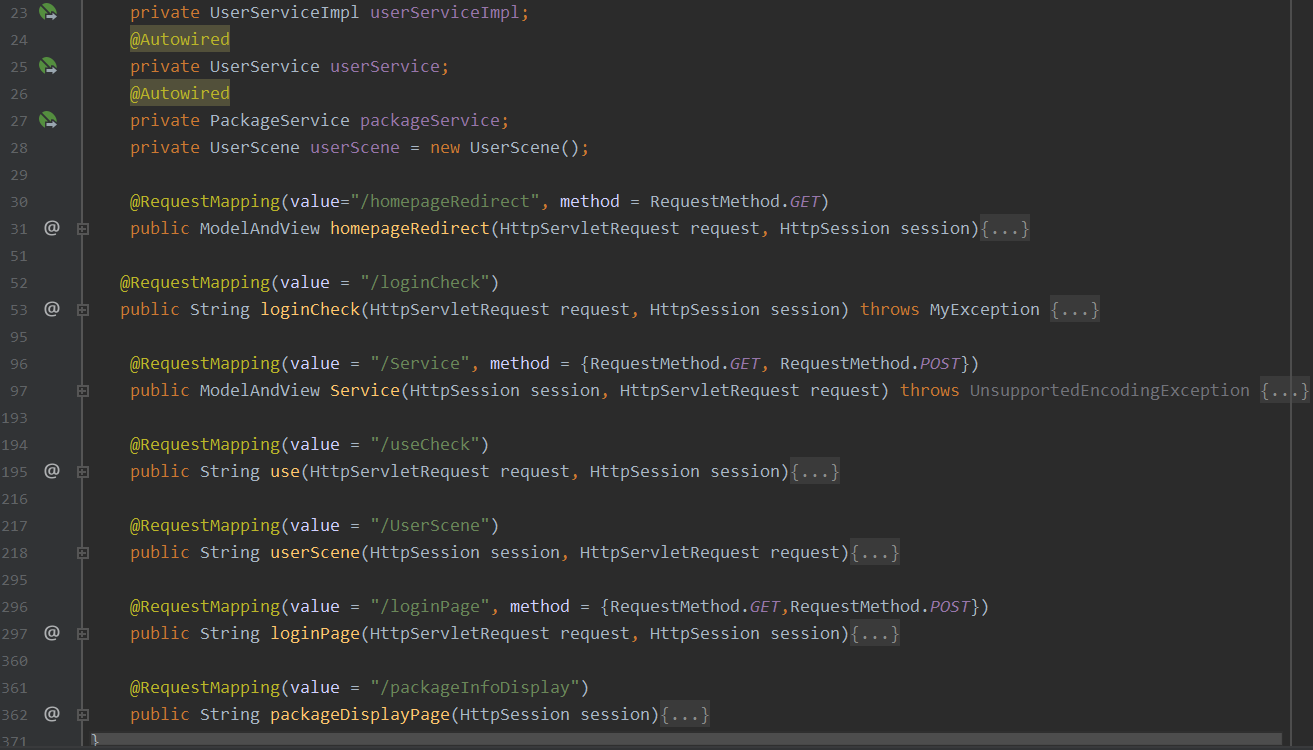


图4-1

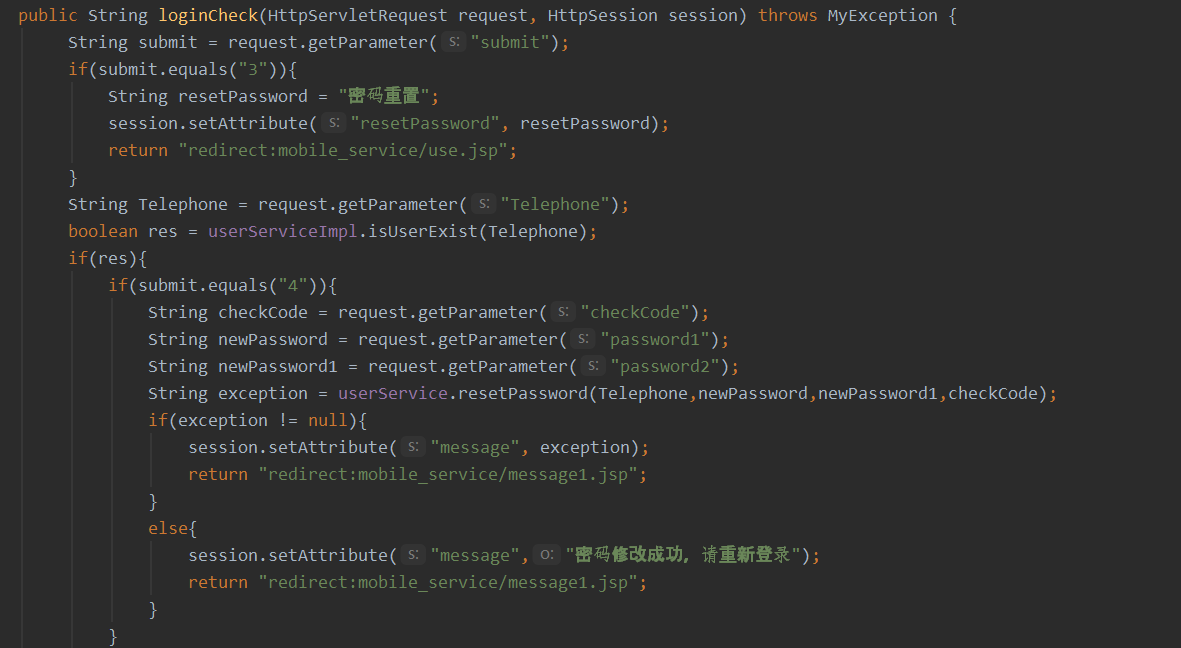
**4.2 Controller和View的交互**

实现Controller和View之间的数据传输花费了不少的时间。因为直接用Model的setAttribute和ModelAndView的setObject都无法便捷的在jsp上获取数据。因此我采用的方法是，当Controller有数据要传到View的时候通过session完成，在View层级获取到这个值并完成一定的工作后，需要手动将session的某个属性的对应值销毁（归零），以防止影响下一步的传输和显示。如果View有数据要传回Controller，我就使用表单提交的方式，在controller中获取request对应属性的值已完成传递。

这种方法简单有效，同时在jsp界面中可以借助Java语言进行简单地逻辑判定，从而使一个界面实现更丰富的功能。

以loginCheck为例，当我的界面有登录、重置和忘记密码等多个按钮时，我通过一个hidden的值来确定被用户按下的是哪一个按钮，从而实现多个功能。而这个值通过request获取。此即图4-2中的submit的用意。当用户按下忘记密码的按钮后，我通过session的属性message向jsp传递指令，使之显示出重置密码的控件。而涉及到与数据库相关的请求或者是业务逻辑相关的判断，我都直接调用了userService和userServiceImpl的函数。这样一来就把业务逻辑、数据库操作、前端逻辑控制有效地切分开。

而当代码内部逻辑完成工作得出结果之后，我们通过“redirect:mobile\_service/\*\*\*.jsp”来实现页面的跳转。值得注意的是，这里的路径对应的是我在webapp文件夹的mobile\_service这个package下真实存在的jsp。而RequestMapping对应的路径未必有真实对应的jsp文件存在。而这个路径能否访问，如何写才能访问，是一个比较棘手的问题，稍后在第六点会详细讲到。



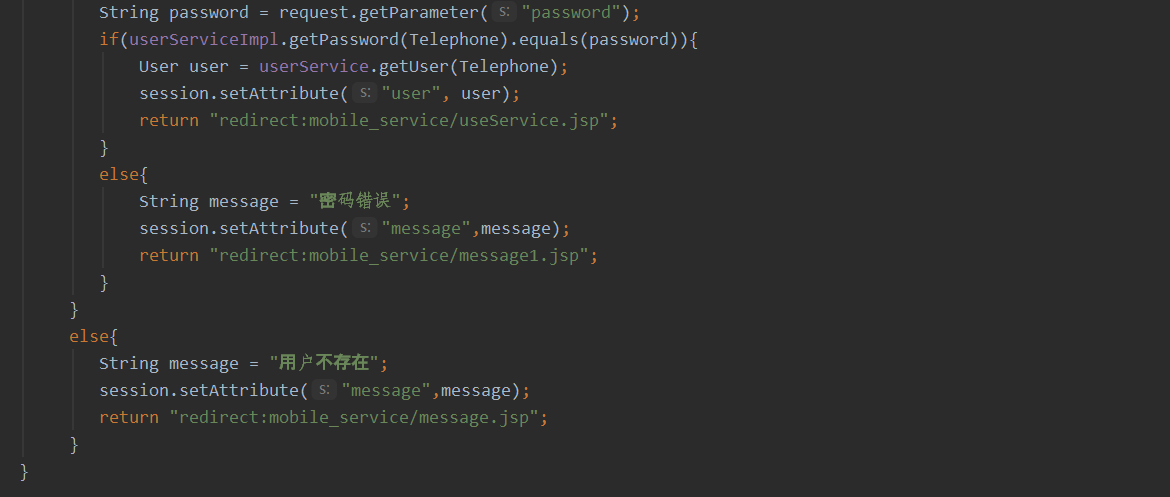


图4-2

1. **前端实现与展示**

**5.1 jsp的设计**

由于对html语言相对陌生，所以使用起来并不习惯。加之本身审美水平较低，所以设计出来的jsp并不美观。唯一可喜可贺的是我在网上学习了一个jsp跳转界面的javascript代码，使网页与网页之间的跳转多了一些生机。

jsp的设计一开始在调用controller传过来的参数时出现了很多问题。尝试了多种方法之后，最终在jsp文件中主要通过编写少量的java代码取出session存储的属性值。另外，我还通过java代码进行页面显示的控制，在需要的时候在同一个页面显示不同的内容，最突出的作用是可以及时的地将代码异常反馈到页面中，进行用户友好的处理，以使程序不停止。

**5.2 具体页面展示**

以下将展示具体的页面实现（部分）。



图5-1



图5-2



图5-3



图5-4



图5-5



图5-6

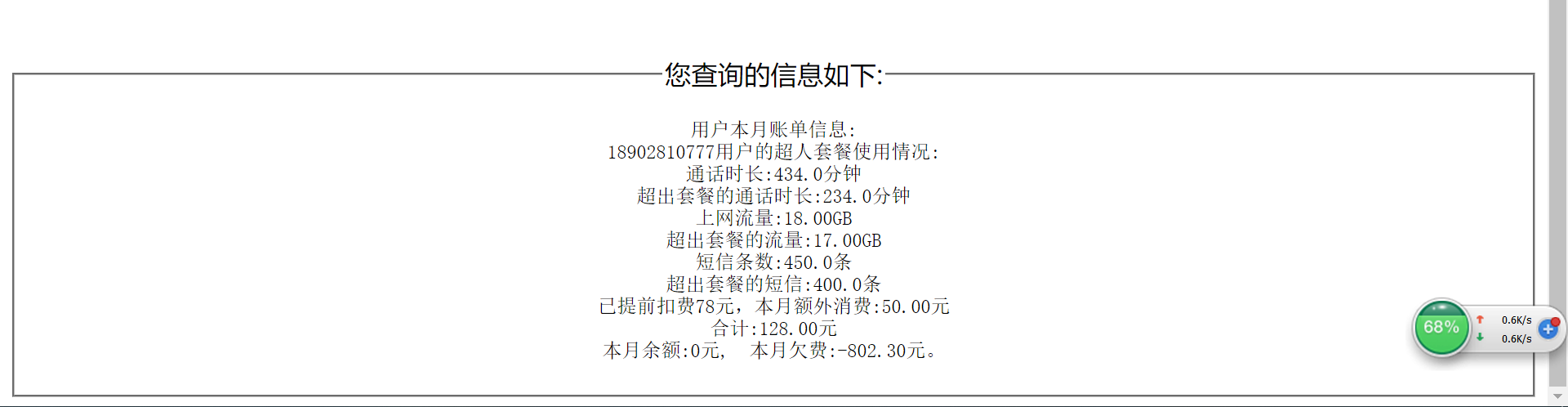


图5-7



图5-8



图5-9



图 5-10

1. **问题和改进**

**6.1 路径访问问题**

之前对于RequestMapping的值和返回的ModelAndView的路径值不太了解，走了许多的弯路。我接下来讲一下自己的理解。RequestMapping实际上是帮我们进行了自动的链接，这个函数相应地对应着一个网址，虽有网址，但无物理实体的页面，只是单纯的映射。而这个返回值所需要符合的格式，实际上应该与spring-mvc的配置文件中InternalResouceViewResolver所设置的前后缀相对应，即mobile\_service/\*\*\*.jsp。我一直将配置文件中的前后缀设置理解成自动补全，实际上它是一种映射，通过这个映射去生成真实物理实体页面所应显示网址，以及关联你实际Java结构目录下的对应位置。这个路径问题一旦解决，应用mvc就变成一件非常愉悦的事情了。

**6.2 框架应用的好处**

应用了SSM框架之后，我对我的代码进行了重新的封装和梳理，不再出现一个简单的类有上千行代码的情况了。最多的都控制在三百行以内。这个过程实际上很好地培养我减少代码冗余的习惯，通过不断逼迫我优化代码来提升能力。

**6.3 框架应用仍存在的问题**

尽管一定程度上代码有了简化，但是在controller类里还是不可避免的夹杂了一些业务逻辑的处理。希望将来可以进一步简化。