Back End Development of An Online Shopping Mall Based on SSM Framework

Jiawei XU

Academic Supervisor: Pierre MARET
Industrial Tutor: Chengcheng XU

30 August 2020

Contents

1 Introduction

The main project of this internship is the development of an online shopping mall based on the SSM framework.

With the development of Internet technology and the promotion of e-commerce competitive ability, more and more stores in traditional fields began to put more attention on online shopping. There are geographical restrictions in the service of customers in traditional fields. And it will be difficult to improve the total number of customers of a store after reaching a certain number that related to the population of this region.

Internet technology provides an opportunity for these stores to display their products in online shopping malls, and geographical restrictions of their business will be greatly reduced. So, they can increase the number of potential customers, increase the sales scale, and then increase the overall profits of stores[1]. In this respect, there are many successful examples, such as Taobao, Amazon and Meituan.

This online shopping mall is aimed at providing service for small stores in the city, such as fast-food restaurants, flower stores, and fruit stores. In the past, these stores attract customers by the way of physical stores and frequent customers' recommendations to new customers. This online shopping mall now provides a cheap and convenient platform for these stores to get more customers at a lower cost and more efficient way.

This online shopping mall could be used in a variety of computer operating systems. In the development process, we use the SpringBoot development framework, MySQL database management system, built-in Tomcat as the website server, and Java and other network programming technologies.

2 Presentation of the Context

2.1 Background of Online Shopping Mall

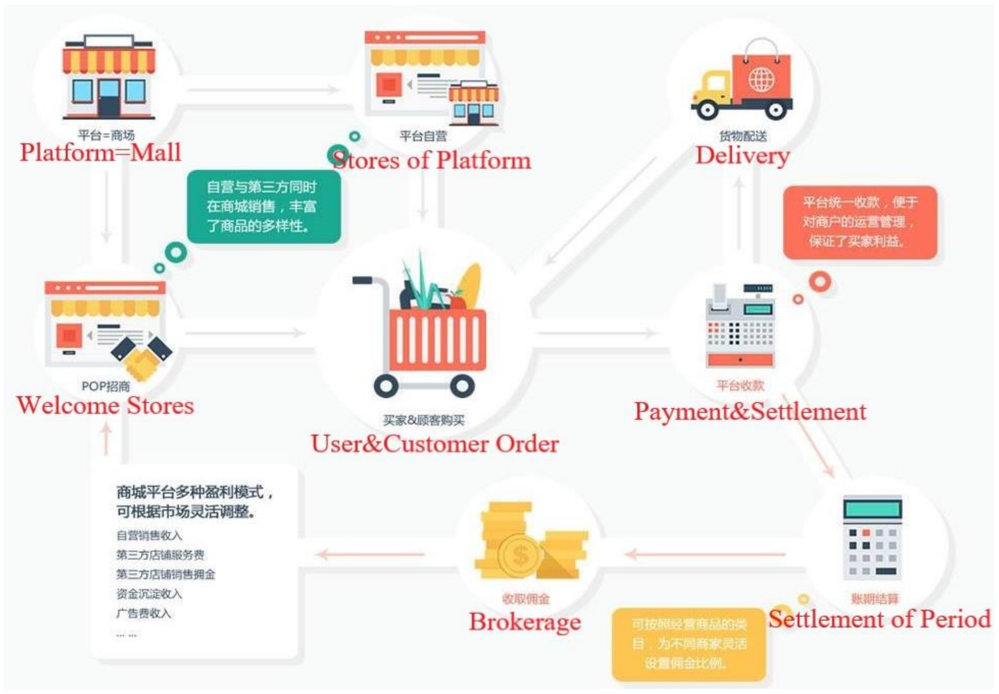


Figure 1: The Map of Online Shopping Mall

After studied the related products of the online shopping mall, we found that Taobao is the largest online shopping mall platform in China with the largest scale, the most variety of goods and the largest geographical coverage. However, many stores in Taobao do not have offline stores, the quality of goods is uncertain, and the timeliness of after-sales is poor. Moreover, Taobao does not have a local business platform for a city at present, and the cost of publicity is relatively expensive, and stores in small cities lack a suitable platform for displaying their own products. In addition, for some timely consumer goods (such as flowers, fruits, seafood, etc.), most customers have more confidence in the quality of offline stores.

O2O business model (online to offline) is a new business model. Customers can browse stores and product catalogs on the Internet, buy directly online, and choose to pick up goods from offline stores or mail home. This mode is different from traditional e-commerce mode. It has more obvious regional attributes[2]. The online shopping mall platform has many offline stores and its own independent logistics system. The advantages of this business model are low initial cost, good shopping experience and good after-sale timeliness.

This online shopping mall has the following competitive advantages:

1. The operators of online shopping malls are closely connected with offline stores to ensure the authenticity of offline stores' qualifications and the reliability and safety of the products they sell;

2. Due to the direct cooperation between the online shopping mall and offline stores, there is no intermediary fee for these stores;

3. Through the membership system of the online shopping mall, consumers could get accumulated consumption coupons or special promotions. These actions could improve their loyalty to this online shopping mall;

4. The online shopping mall could provide online publicity activities for offline stores. This could encourage customers to go to offline stores through the positioning function of the application, and improve the quantity of customer access to offline stores;

5. This online shopping mall has its independent logistics system with high distribution efficiency and lower distribution cost than most regional platforms.

2.2 Analysis of Demand

1. Tourist users

Tourist users are those who have not registered and logged in. As a user, tourist users can open and browse the page information of the website, view the product information by category, search the items with keywords, view the introduction and price of the products and other information. But tourists can't add products to the shopping cart, settle command of goods and accomplish command of goods.

2. Ordinary users

The ordinary user role has all operation rights of the tourist user role. In addition, ordinary users can log in to the space of the user, perfect and modify personal information. They could add products to the shopping cart, settle the command of goods and accomplish command of goods. They also have some other rights[6].

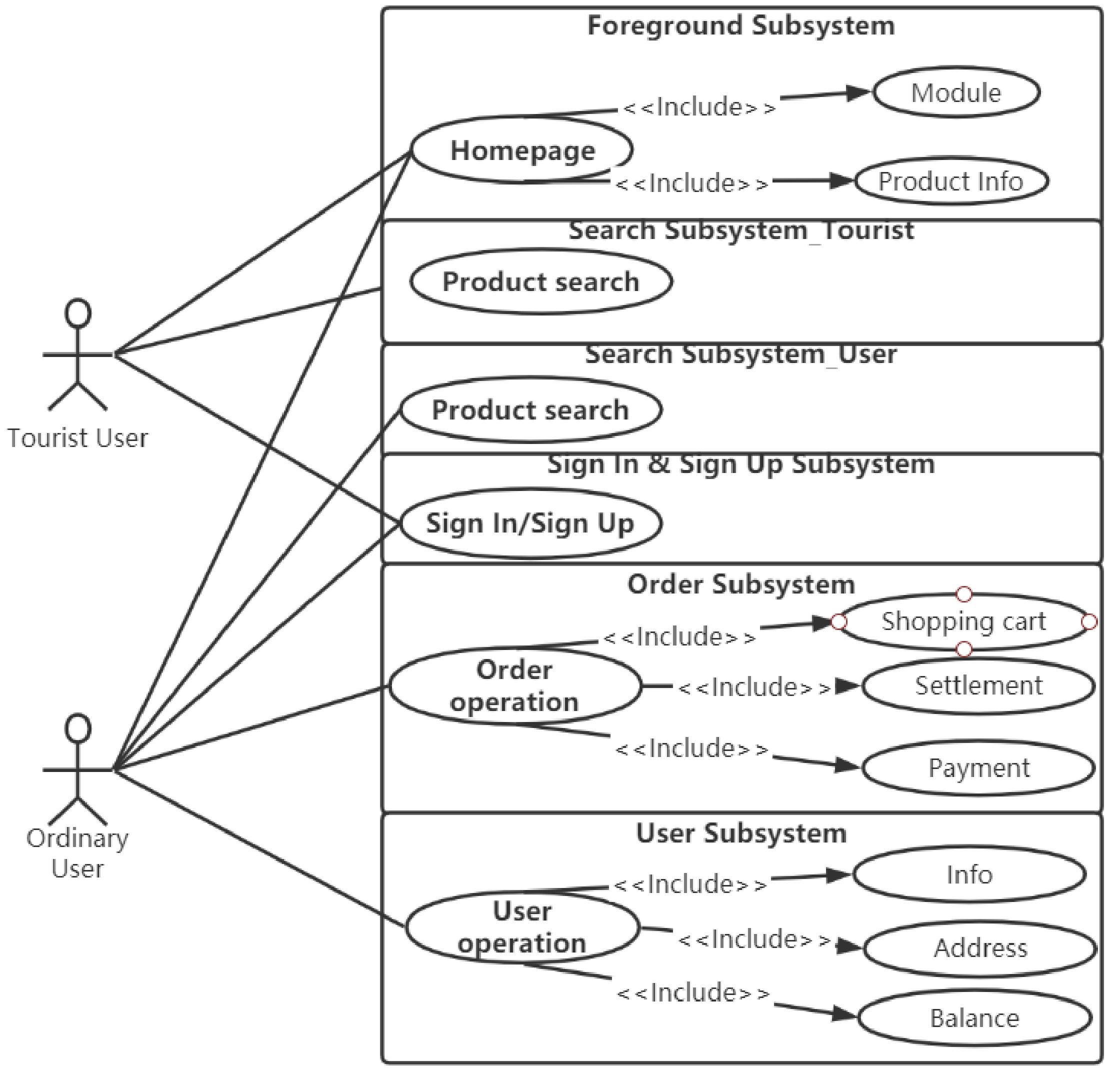


Figure 2: Use Case Diagram of User

3. Administrator of Goods and Order

The administrator of goods and order role has all operation rights of the management of goods and order. As an administrator, he can query the information of goods and orders, create new goods, update information of goods, generate new order and update or cancel order.

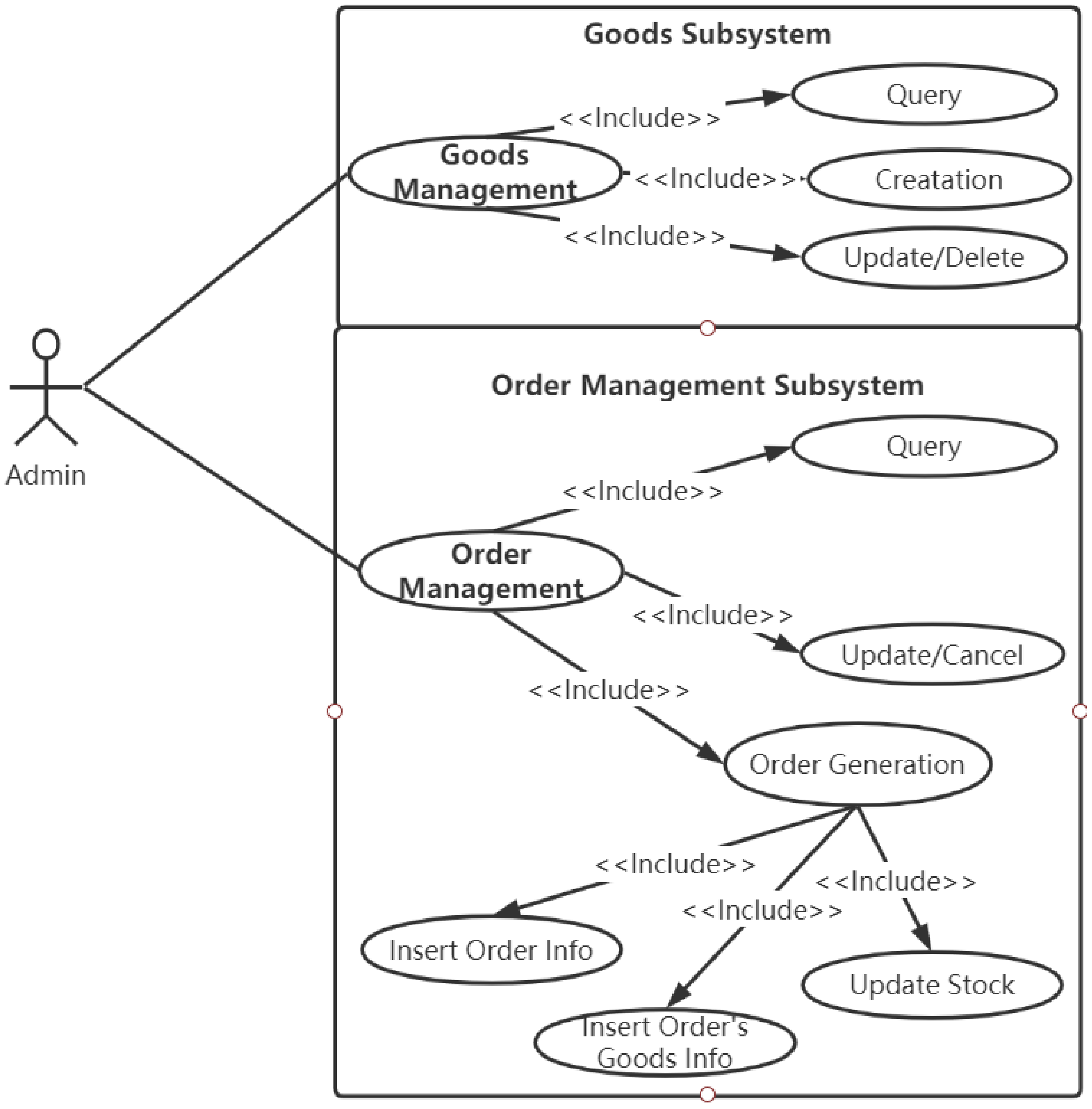


Figure 3: Use Case Diagram of Administrator

3 Presentation of Problem

There are many limitations to be considered in the planning and design of the architecture of this online shopping mall[8]: Scope and size of the business of this online shopping mall; Development cost and time cost of the website; Low cost of maintenance in the later period; To meet the usage habits of stores and customers.

The ideal online shopping mall architecture should not only adapt to the different needs of different users but also reduce the expansion cost and maintenance cost as much as possible in the future when there is an expansion of scale. Therefore, in the architecture design of this online shopping mall, we need to be able to solve all the needs of users. At the same time, this online shopping mall should have a high level of loose coupling, easy to maintain, have good scalability, and must reduce the cost of hardware requirements[3].

Stage1: A single application architecture is used

In the early stage of development, the general number of users of the website is not large. The number of visitors is not large. The amount of data is not large. And all the programs and software could run on a single computer. At this time, we could use open-source framework Maven + Spring + SpringMVC + Mybatis. Select MySQL management system to store data, and then connect and operate the database through JDBC.

All the above software including database and application program are loaded on the same computer. When all the applications running on the computer, a small online shopping mall is working. At this time, the architecture of the system is as follows:

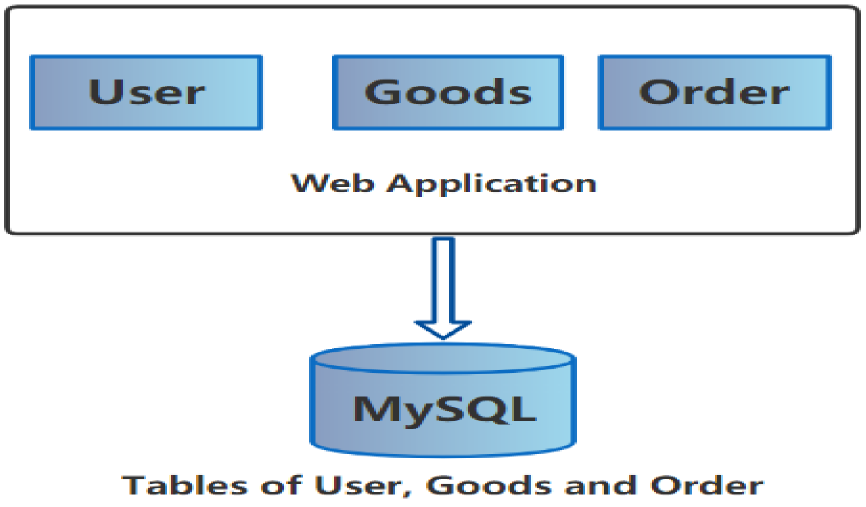


Figure 4: The Architecture of Single Computer

In the future, this online shopping mall goes online, it is predicted that within three years, it can serve around 20 small and medium-sized cities, with more than 5000 stores, 300000 registered users and 20000 times daily visit on the Internet. Such a large user group will inevitably produce a big amount of concurrent access. Simple single architecture system, if only rely on the support of hardware, it is difficult to load such high demand. For operators of the online shopping mall, relying on continuous hardware improvement to meet user access needs, development and construction costs are too high, which is not cost-effective in the long term.:

Stage 2: Separation of application server and database

Stage3: Application of server cluster

4 Existing Solutions

4.1 JSJ: JSP+Servlet+JavaBean

1. JavaBean: as a model, it could not only encapsulate service data as a data model but also contain service operations of applications as a service logic model. Among them, the data model is used to store or transfer service data. After receiving the model update request from the controller, the service logic model performs specific service logic processing and then returns the corresponding execution results.

2. JSP: Java Server Pages, as the presentation layer, it is responsible for providing the page to show the data to the user, providing the corresponding form for the user's request, and at the appropriate time (click the button) sending a request to the controller to request the model to update.
3. Servlet: as a controller, it is used to receive requests submitted by users, then get data in the request, transform them into data models needed by service models, then update service models corresponding to service models, and select the view to be returned according to the results of service execution.

4.2 SSH: Struts+Spring+Hibernate

1. Struts: an MVC framework based on the Sun J2EE platform, which is mainly implemented by the servlet and JSP technology. As struts can fully meet the needs of application development, it is easy to use, agile and rapid and has attracted much attention in the past year. Struts integrate servlets, JSPS, custom tags and message resources into a unified framework. When developers use them to develop, they don't need to code to implement a full set of MVC mode, which greatly saves time. Therefore, Struts is a very powerful application framework.

2. Spring: a powerful framework that solves many common problems in J2EE development. Spring provides a consistent way to manage service objects. Spring's architecture is based on the inversion of control container using JavaBean properties. Spring is unique in using IOC containers as a complete solution for building all architectural layers. Spring provides a unique data access abstraction, including a simple and efficient JDBC framework, which greatly improves efficiency and reduces possible errors. Spring's data access architecture also integrates hibernate and other O/R mapping solutions. Spring also provides a unique transaction management abstraction, which can provide a consistent programming model in various underlying transaction management technologies, such as JTA or JDBC transactions. Spring provides an AOP framework written in the standard Java language, which provides POJOs with declarative transaction

management and other enterprise transactions - if you need them - as well as your own aspects. Spring also provides a powerful and flexible MVC web framework that can be integrated with IOC containers.

3. Hibernate: an open-source object-relational mapping framework, which encapsulates JDBC with very lightweight objects, so that Java programmers can use object programming thinking to manipulate the database as they like. Hibernate can be used in any situation where JDBC is used. It can be used not only in Java client programs but also in servlet/JSP Web applications. The most revolutionary thing is that hibernate can replace CMP in the J2EE architecture of EJB application to complete the task of data persistence.

4.3 SSM: Spring+SpringMVC+Mybatis

1. Spring: As the above part.
2. SpringMVC: it belongs to the derivative module of the Spring framework. SpringMVC calls the three major parts of MVC: Controller, Model and View through the front-end controller (also called Central Scheduler), DispatcherServlet. In this way, each component of MVC is only coupled with Dispatcher Servlet, and each component runs independently, which greatly reduces the coupling of the program.
3. Mybatis: a collection of concepts and methods of multiple operational relational databases. It is a powerful data access tool and solution[4]. A Java-based persistence framework. The persistence layer framework provided by iBATIS includes SQL Maps and Data Access Objects (DAO) Mybatis, which eliminates almost all the manual settings of JDBC code and parameters and the retrieval of result sets. Mybatis uses simple XML or annotations for configuration and raw mapping, mapping the interface and POJOs (Plain Old Java Objects) of Java to records in the database.

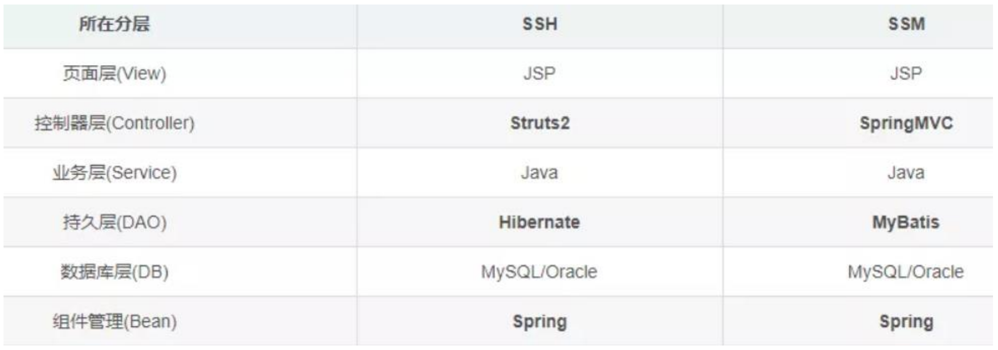
5 Selection of a Solution

5.1 Brief conlcusion of JSJ

In short, the development mode of JSP + servlet + JavaBean needs to write a lot of repetitive code, such as the fixed doGet () method. And its control jump is not flexible, often a problem dealing with Dao requires two java files. When MVC mode is used in the development, there is a great degree of coupling, and later maintenance will be very difficult.

The SSH or SSM framework can make developers pay more attention to the code development of the Service layer. It saves a lot of time and energy for low-level and repetitive code writing. The following will mainly compare the advantages and disadvantages of the two development modes and which is more suitable for this project.

5.2 Final：SSM V.S SSH



所在分层	SSH	SSM
页面层(View)	JSP	JSP
控制器层(Controller)	**Struts2**	**SpringMVC**
业务层(Service)	Java	Java
持久层(DAO)	**Hibernate**	**MyBatis**
数据库层(DB)	MySQL/Oracle	MySQL/Oracle
组件管理(Bean)	**Spring**	**Spring**

Figure 5: Layers of SSH and SSM

1. Differences between Struts2 and SpringMVC:

a) In SpringMVC, requests are for methods, that is, a method corresponds to a request. This belongs to method interception. The requested data method is not shared with others; While in struts2, the request is for to an action class, that is, an action class corresponds to a request, and the requested data is shared;

b) The entry of SpringMVC is a servlet front-end controller (Dispatcher servlet), but the entry of struts 2 is a filter;

c) Compared with struts 2, the configuration files of SpringMVC are less and easier to use. This can be beneficial to the speed of software development.

2. Differences between Hibernate and Mybatis:

a) Hibernate is O/R relational type, which completes the mapping between database tables and persistent classes. Mybatis is for SQL mapping. After hibernate encapsulates the database, it calls the corresponding database operation statement of SQL. But Mybatis uses the original database operation statement;

b) For advanced queries. Mybatis needs to write SQL statements and result maps manually. But Hibernate has a good mapping mechanism. Developers don't need to care about the generation of SQL and result mapping, so they can focus on service processes;

c) Hibernate is more difficult to optimize than Mybatis, and it is more difficult to master Hibernate than Mybatis. But the Hibernate database is more portable that of Mybatis database. In Mybatis, different databases need different statements of SQL.

In short, SpringMVC could make the design of the online shopping mall conform to the way of program development and the design principle of high cohesion and low coupling[7].

SSM is becoming more and more lightweight in configuration, and take the most advantage of annotation development. It also making ORM more flexible and optimization of SQL becoming easier. Using SSH needs to pay more attention to configuration development, in which Hibernate is more object-oriented and more automatic in the data maintenance of addition, deletion, and query. However, the optimization of SQL in Hibernate is weak, and the learning threshold is higher.

Compared with Hibernate, Mybatis is a semi-automatic framework. Hibernate is a fully automatic framework, unable to maintain SQL directly. Mybatis is very flexible in writing statements of SQL. While Hibernate is not good at this part, it will be more troublesome than Mybatis. In this project, using Mybatis is suitable for the project of the online shopping mall, and Hibernate is suitable for the stable project.

6 Implementation of the Solution

In the framework of Spring, the most basic parts are Controller layer, Service layer and Dao layer.

The Controller layer mainly views the service functions and displays the results of the implemented functions. The SpringMVC framework is the framework to implement the Controller layer. The framework finds the corresponding controller through the dispatch servlet and submits the user's request to the controller. Then the controller calls the service logic component and returns to ModelAndView. Then call the view parser to find the corresponding view of ModelAndView, and display the result on the terminal.

In Spring, the Service layer is mainly a layer marked by "@service" annotation. It includes the development of the implementation class of the service interface. It provides methods could be called by the Controller layer. It does not operate the database directly, but will process the data in some way, such as condition judgment and data filtering.

Dao layer is Mapper layer. It performs persistent operations on the database. Its method is aimed at database operation, and its main methods are adding, deleting, updating and querying. The specific implementation of the interface is in the mapper.xml file.

There are two ways to integrate Mybatis: annotation and configuration of XML. The SQL statement is written in the annotation way directly, which avoids the configuration of the XML file. Compare with the mapping method of configuration of the XML file, the annotation way is obviously more concise, but it is not flexible. The way of the configuration of XML is more maintainable. Therefore, we used the second way.

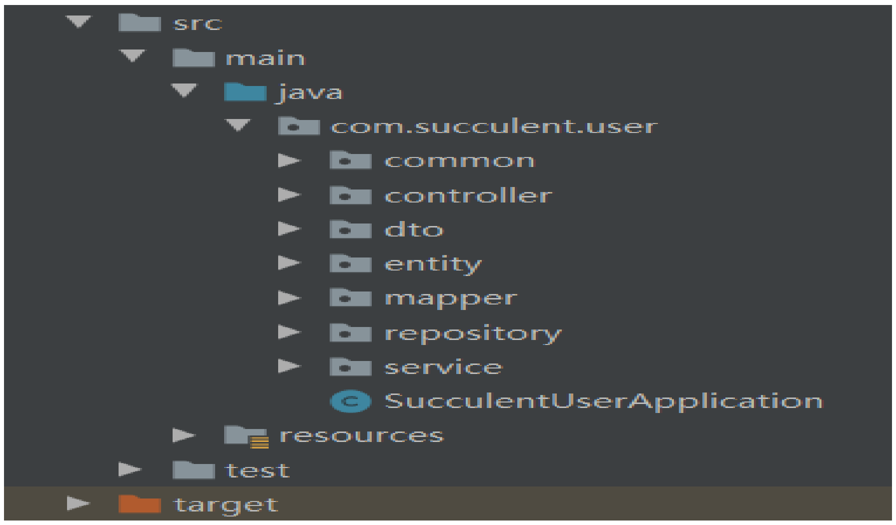


Figure 6: Structure of Project

Take login for example. When users access the page of the online shopping mall, the data first enter the Controller layer through the API interface. Then the system calls the HTML file through the Controller layer to display the login page.

Next, the user enters the mobile number and password. After the user clicks "login", the user name and password entered in the login page will be submitted. The foreground page submits the user information (mobile number and password) to the Controller layer according to the pre-written interface. After receiving the information from the foreground, the Controller layer transfers the user's information to the Service layer's method. At the same time, the system also accesses the Dao layer through the Service layer.

The Dao layer goes to the database to access the data in the database (mobile number and password in the database), and then return to the Service layer. At this point, data from two different sources are compared. If the two sets of data are the same, the login is successful. The Service layer returns the information to the Controller layer to inform it that the authentication has passed, and the Controller layer calls the page that has successfully logged in, so the user can carry out the next operation; if the matching fails, the Service layer informs the Controller layer that the authentication has failed, and the Controller layer calls the page that has failed to log in or returns to the login in page.

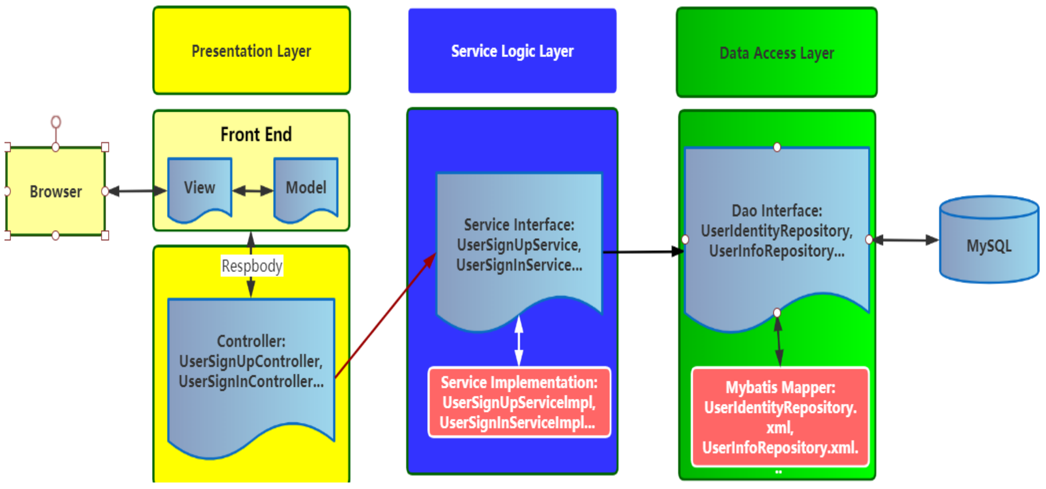

Presentation Layer
Service Logic Layer
Data Access Layer
Browser
Front End
View
Model
Respbody
Controller: UserSignUpController, UserSignInController...
Service Interface: UserSignUpService, UserSignInService...
Service Implementation: UserSignUpServiceImpl, UserSignInServiceImpl...
Dao Interface: UserIdentityRepository, UserInfoRepository...
Mybatis Mapper: UserIdentityRepository.xml, UserInfoRepository.xml..
MySQL

Figure 7: The Diagram of Logic of Web Application

6.1 Configuration of Pom.xml

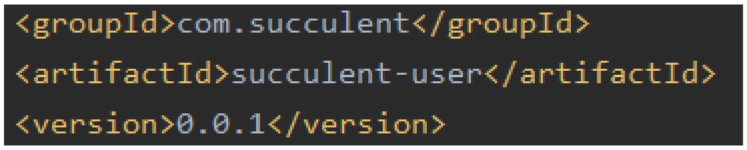
The configuration used by Maven to build a project is specified in pom.xml. This file defines the Project Object Model for Maven.

Some parts of this file:

1. groupId: defines the actual project to which the current Maven project belongs.

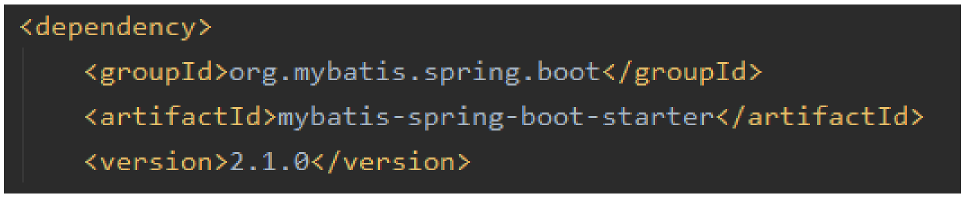
artifactId: defines the Maven project of the actual project.

version: defines the current version of the Maven project.



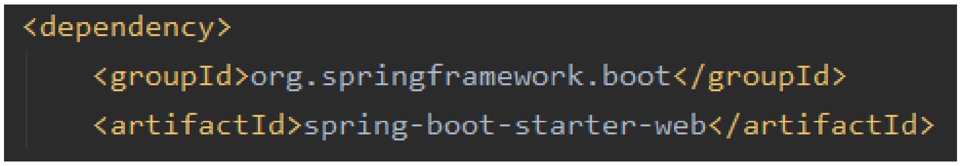
```
<groupId>com.succulent</groupId>
<artifactId>succulent-user</artifactId>
<version>0.0.1</version>
```

2. Add modules for supporting Mybatis. Add the corresponding dependency in the pom.xml file.

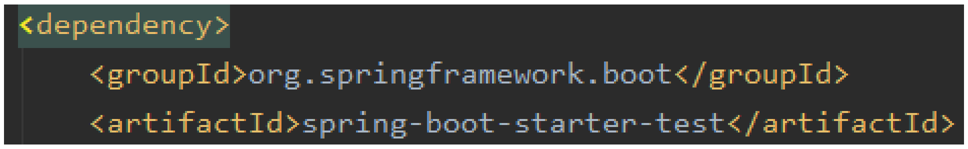


```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.0</version>
```

3. There are two default modules in this file: Spring boot starter: core module, including auto-configuration support, log and YAML; Spring boot starter test: test module, including JUnit, Hamcrest and Mockito[5].



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

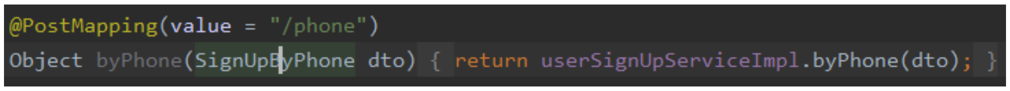


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
```

6.2 User Sign Up Function

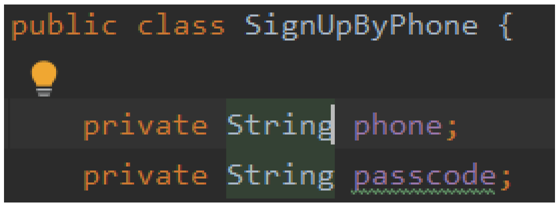
UserSignUpController: This class is responsible for the logic of user sign up function: this class specifically performs operations on create new user account. There is 1 method: byPhone.

byPhone() uses the post request method, and it is responsible for creating the user account.



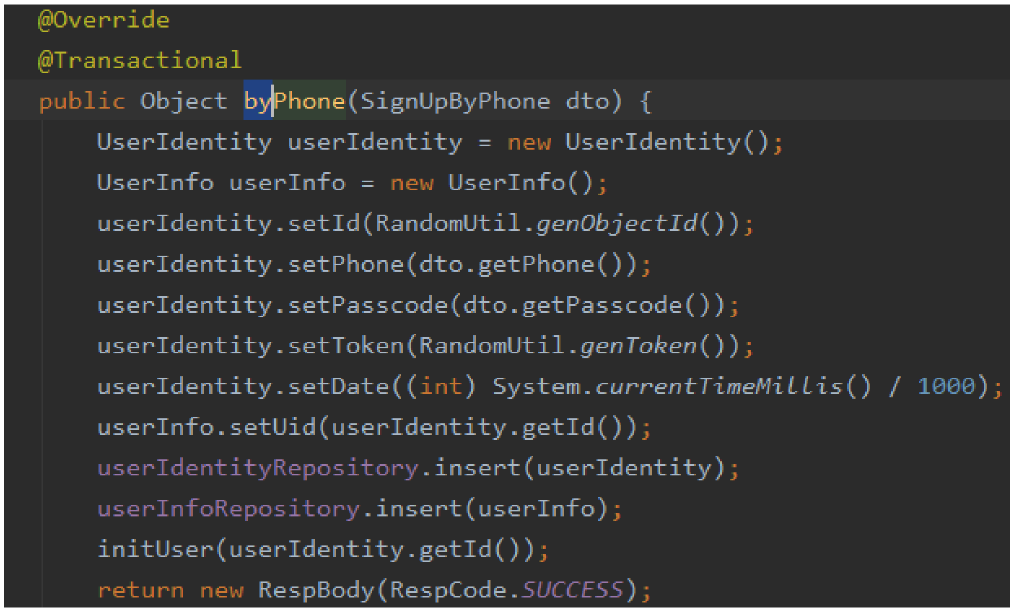
```
@PostMapping(value = "/phone")
Object byPhone(SignUpByPhone dto) { return userSignUpServiceImpl.byPhone(dto); }
```

This byPhone() method receives the post request to create the user account, and can carry parameters such as phone, passcode, as shown in the following figure:



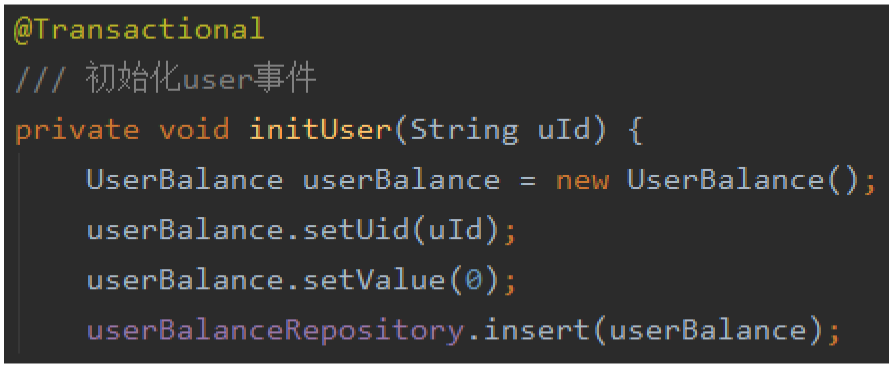
```java
public class SignUpByPhone {

    private String phone;
    private String passcode;
```

And it calls the byPhone() method of the implementation class userSignUpServiceImpl of the interface userSignUpService for further processing.



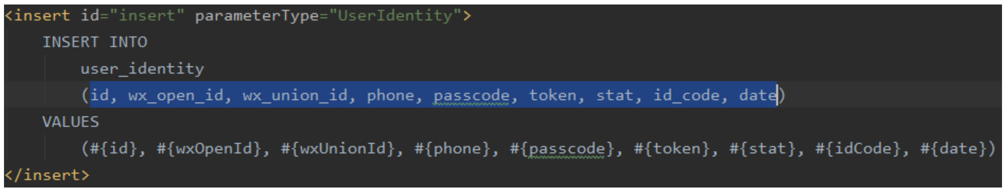
```java
@Override
@Transactional
public Object byPhone(SignUpByPhone dto) {
    UserIdentity userIdentity = new UserIdentity();
    UserInfo userInfo = new UserInfo();
    userIdentity.setId(RandomUtil.genObjectId());
    userIdentity.setPhone(dto.getPhone());
    userIdentity.setPasscode(dto.getPasscode());
    userIdentity.setToken(RandomUtil.genToken());
    userIdentity.setDate((int) System.currentTimeMillis() / 1000);
    userInfo.setUid(userIdentity.getId());
    userIdentityRepository.insert(userIdentity);
    userInfoRepository.insert(userInfo);
    initUser(userIdentity.getId());
    return new RespBody(RespCode.SUCCESS);
```

This byPhone() method configs initial information of user account such as generate random Id of user, generate random Token for furthur usage, set sign up date and initialize the balance of this user.

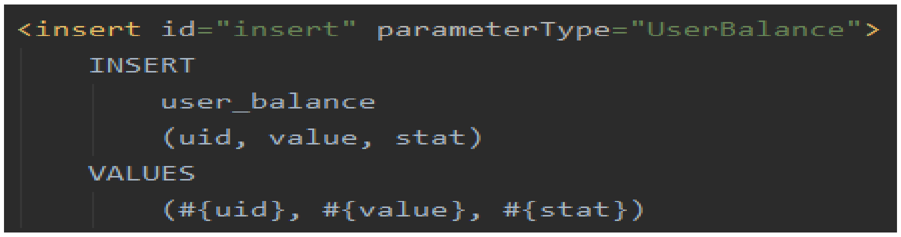


```java
@Transactional
/// 初始化user事件
private void initUser(String uId) {
    UserBalance userBalance = new UserBalance();
    userBalance.setUid(uId);
    userBalance.setValue(0);
    userBalanceRepository.insert(userBalance);
```

Then it forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:

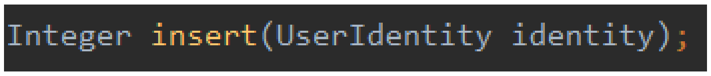


```
<insert id="insert" parameterType="UserIdentity">
    INSERT INTO
        user_identity
        (id, wx_open_id, wx_union_id, phone, passcode, token, stat, id_code, date)
    VALUES
        (#{id}, #{wxOpenId}, #{wxUnionId}, #{phone}, #{passcode}, #{token}, #{stat}, #{idCode}, #{date})
</insert>
```

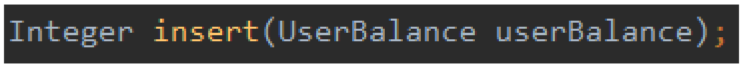


```
<insert id="insert" parameterType="UserBalance">
    INSERT
        user_balance
        (uid, value, stat)
    VALUES
        (#{uid}, #{value}, #{stat})
```

If the interactions with the database are successful, the success message are returned to the insert() method of the corresponding UserIdentityRepository and insert() method of the corresponding UserBalanceRepository:



```
Integer insert(UserIdentity identity);
```

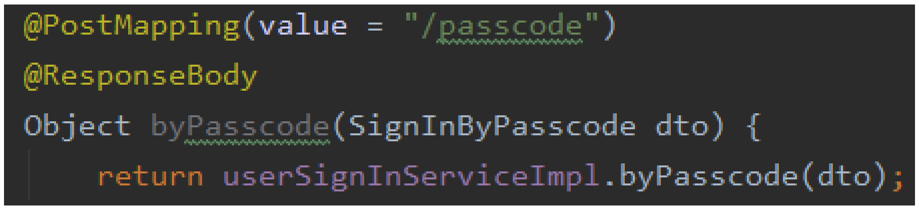


```
Integer insert(UserBalance userBalance);
```

6.3 User Sign In Function

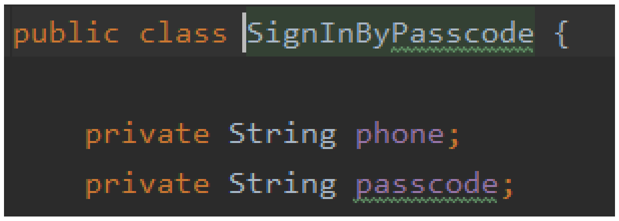
UserSignInController: This class is responsible for the logic of user sign in function: this class specifically performs operations on sign in the user account. There is 1 method: byPasscode.

byPasscode() uses the post request method, and it is responsible for sign in the user account.



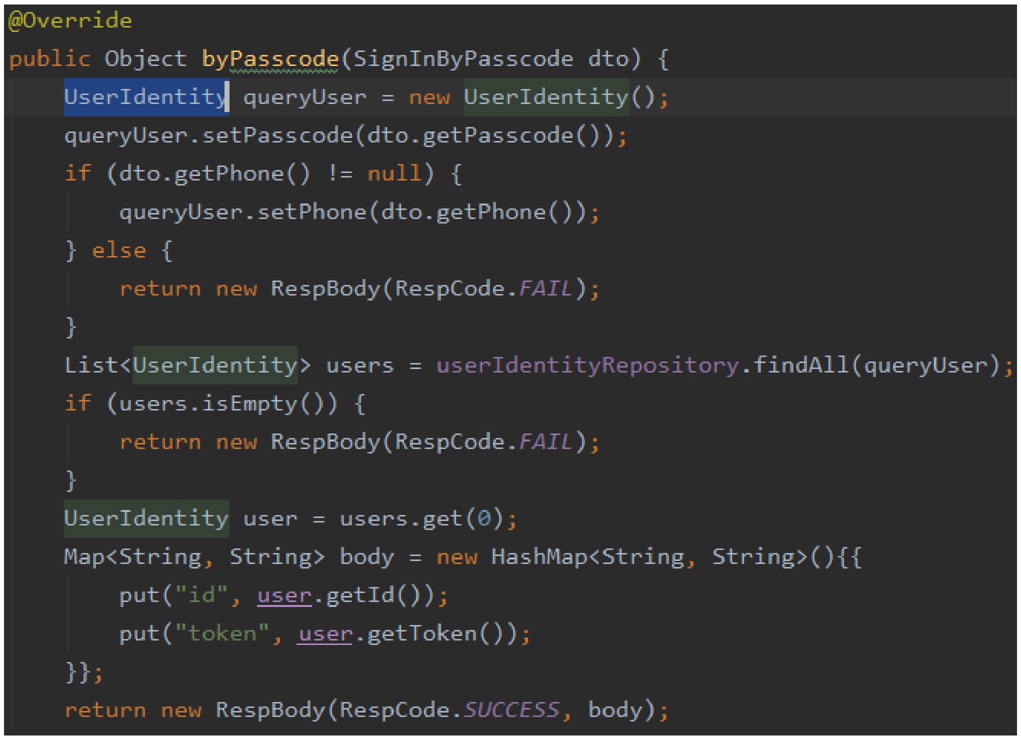
```
@PostMapping(value = "/passcode")
@ResponseBody
Object byPasscode(SignInByPasscode dto) {
    return userSignInServiceImpl.byPasscode(dto);
```

This byPasscode() method receives the post request to create the user account, and can carry parameters such as phone, passcode, as shown in the following figure:



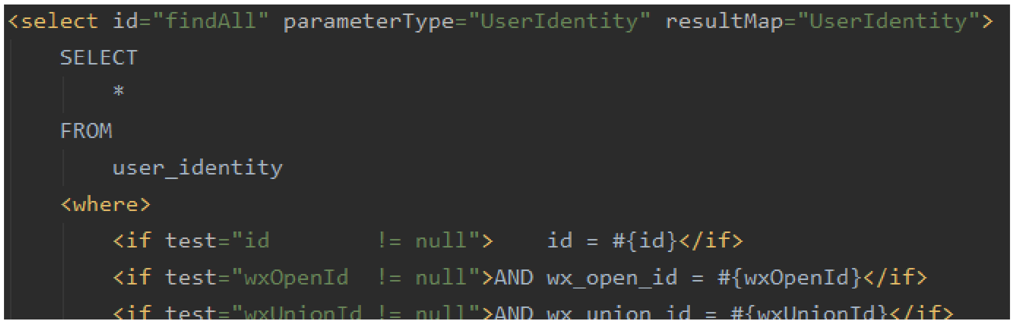
```
public class SignInByPasscode {

    private String phone;
    private String passcode;
```

And it calls the byPasscode() method of the implementation class userSignInServiceImpl of the interface userSignInService for further processing.



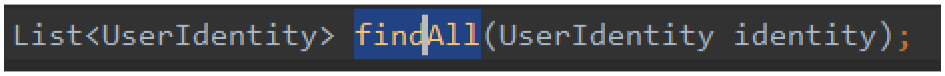
```java
@Override
public Object byPasscode(SignInByPasscode dto) {
    UserIdentity queryUser = new UserIdentity();
    queryUser.setPasscode(dto.getPasscode());
    if (dto.getPhone() != null) {
        queryUser.setPhone(dto.getPhone());
    } else {
        return new RespBody(RespCode.FAIL);
    }
    List<UserIdentity> users = userIdentityRepository.findAll(queryUser);
    if (users.isEmpty()) {
        return new RespBody(RespCode.FAIL);
    }
    UserIdentity user = users.get(0);
    Map<String, String> body = new HashMap<String, String>(){{
        put("id", user.getId());
        put("token", user.getToken());
    }};
    return new RespBody(RespCode.SUCCESS, body);
```

This byPasscode() method verify user account and his passcode. It forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



```xml
<select id="findAll" parameterType="UserIdentity" resultMap="UserIdentity">
    SELECT
        *
    FROM
        user_identity
    <where>
        <if test="id         != null">    id = #{id}</if>
        <if test="wxOpenId   != null">AND wx_open_id = #{wxOpenId}</if>
        <if test="wxUnionId != null">AND wx_union_id = #{wxUnionId}</if>
```

If the interaction with the database is successful, the resultMap object is returned to the findAll() method of the corresponding UserIdentityRepository:



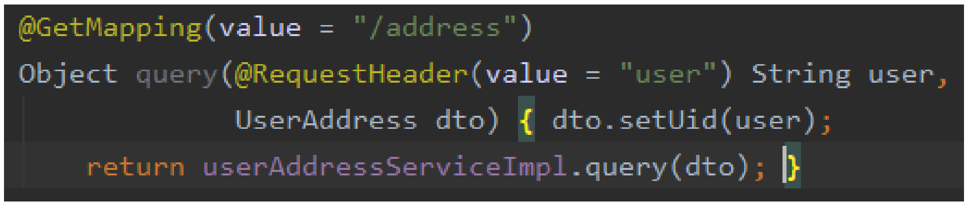
```java
List<UserIdentity> findAll(UserIdentity identity);
```

6.4 Management of User’s Address Information

UserAddressController: This class is responsible for the logic of user address information management: this class specifically performs operations on user addresses. There are 4 methods in total, namely query, findById, insert and updateById.

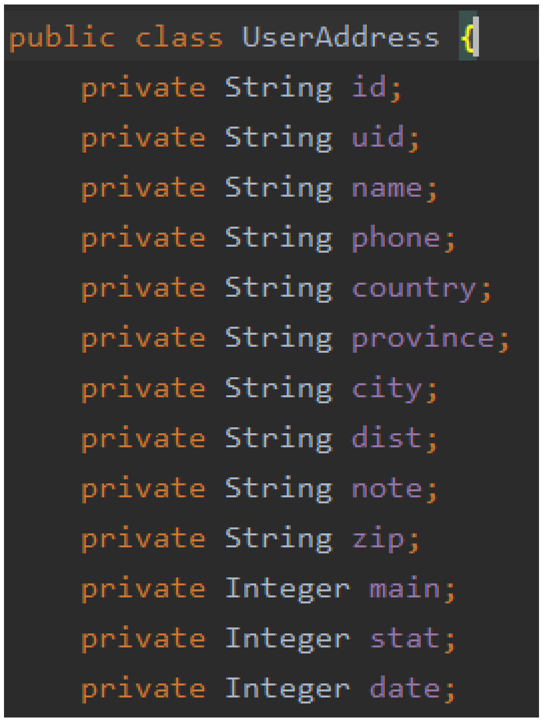
1. **query() Method:**

   query() uses the get request method, and it is responsible for querying the user's address information.



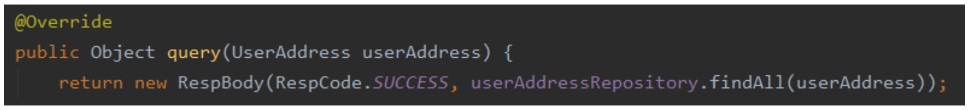
```java
@GetMapping(value = "/address")
Object query(@RequestHeader(value = "user") String user,
                 UserAddress dto) { dto.setUid(user);
    return userAddressServiceImpl.query(dto); }
```

   This query() method receives the get request to query the user address, and can carry parameters such as id, uid, name, phone, zip, note, main, stat, date, country, province, city, dist, as shown in the following figure:



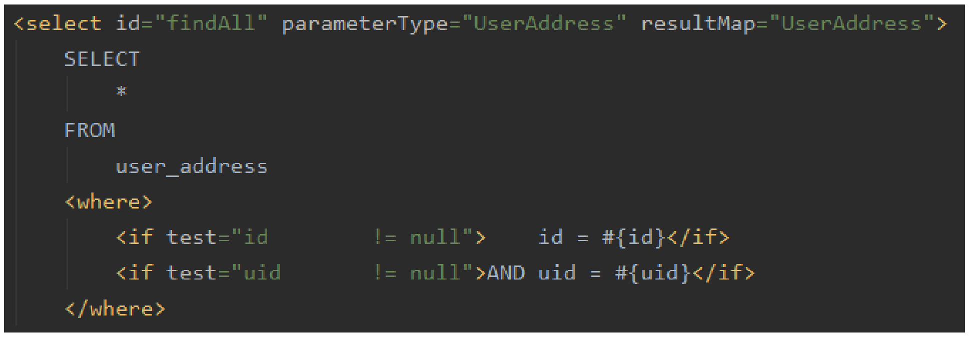
```java
public class UserAddress {
    private String id;
    private String uid;
    private String name;
    private String phone;
    private String country;
    private String province;
    private String city;
    private String dist;
    private String note;
    private String zip;
    private Integer main;
    private Integer stat;
    private Integer date;
```

   And it calls the query() method of the implementation class userAddressServiceImpl of the interface userAddressService for further processing.



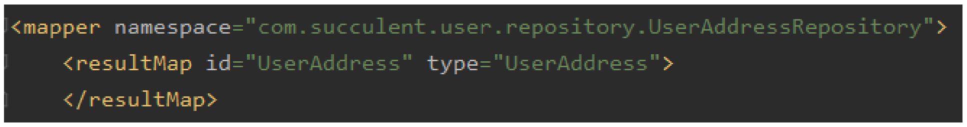
```java
@Override
public Object query(UserAddress userAddress) {
    return new RespBody(RespCode.SUCCESS, userAddressRepository.findAll(userAddress));
```

   This query() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:

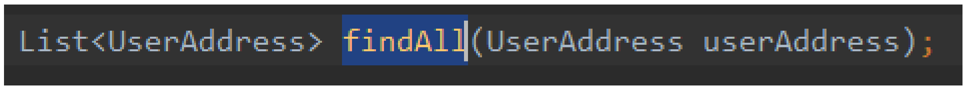


```
<select id="findAll" parameterType="UserAddress" resultMap="UserAddress">
    SELECT
        *
    FROM
        user_address
    <where>
        <if test="id        != null">    id = #{id}</if>
        <if test="uid       != null">AND uid = #{uid}</if>
    </where>
```

If the interaction with the database is successful, the resultMap object is returned to the findAll() method of the corresponding UserAddressRepository:



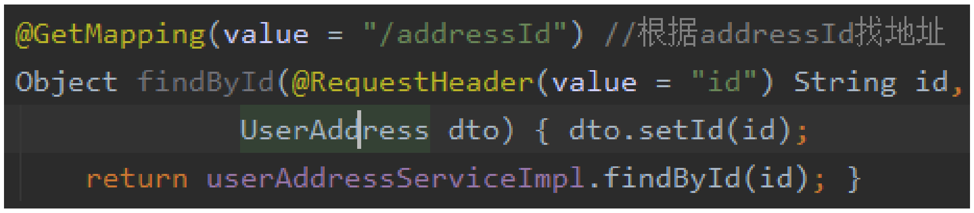
```
<mapper namespace="com.succulent.user.repository.UserAddressRepository">
    <resultMap id="UserAddress" type="UserAddress">
    </resultMap>
```



```
List<UserAddress> findAll(UserAddress userAddress);
```

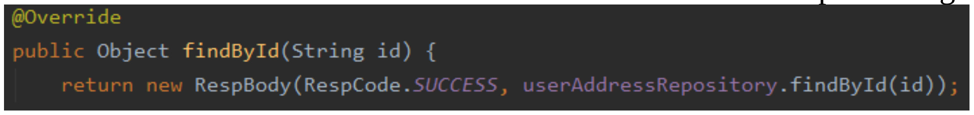
2. **findById() Method:**

findById() uses the Get request method, and it is responsible for finding the address information by Id of this address.



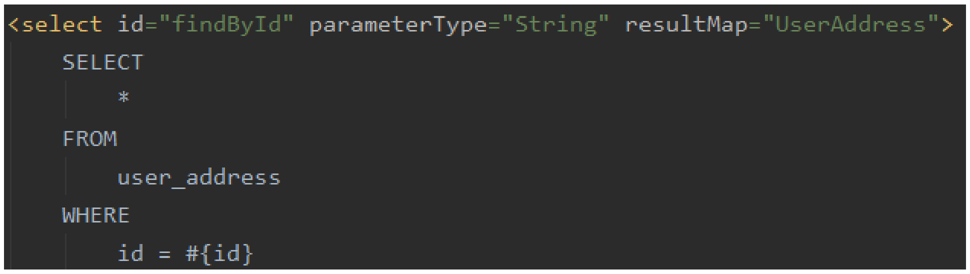
```
@GetMapping(value = "/addressId") //根据addressId找地址
Object findById(@RequestHeader(value = "id") String id,
                UserAddress dto) { dto.setId(id);
    return userAddressServiceImpl.findById(id); }
```

This insert() method receives the Get request to find the address information by Id of this address, and can carry parameter of id. And it calls the findById() method of the implementation class userAddressServiceImpl of the interface userAddressService for further processing.



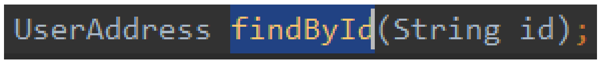
```
@Override
public Object findById(String id) {
    return new RespBody(RespCode.SUCCESS, userAddressRepository.findById(id));
```

This findById() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<select id="findById" parameterType="String" resultMap="UserAddress">
    SELECT
        *
    FROM
        user_address
    WHERE
        id = #{id}
```

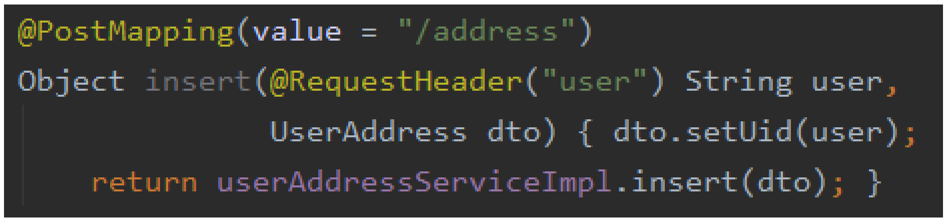
If the interaction with the database is successful, the resultMap object is returned to the findById() method of the corresponding UserAddressRepository:



```
UserAddress findById(String id);
```

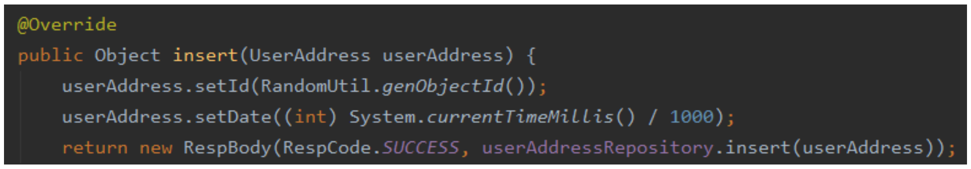
3. insert() Method:

insert() uses the Post request method, and it is responsible for inserting the new user's address information.



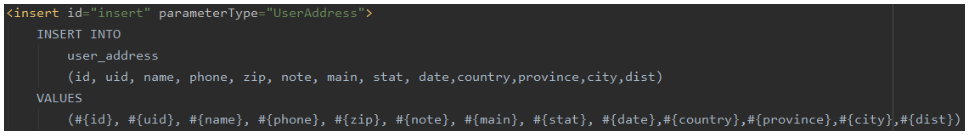
```
@PostMapping(value = "/address")
Object insert(@RequestHeader("user") String user,
              UserAddress dto) { dto.setUid(user);
   return userAddressServiceImpl.insert(dto); }
```

This insert() method receives the post request to insert the new user address, and can carry parameters such as id, uid, name, phone, zip, note, main, stat, date, country, province, city, dist, as shown in the above figure of UserAddress class. And it calls the insert() method of the implementation class userAddressServiceImpl of the interface userAddressService for further processing.



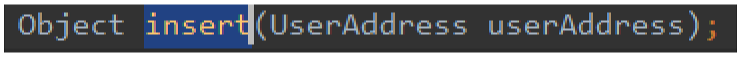
```
@Override
public Object insert(UserAddress userAddress) {
    userAddress.setId(RandomUtil.genObjectId());
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    return new RespBody(RespCode.SUCCESS, userAddressRepository.insert(userAddress));
```

This insert() method forwards the data to the dao layer to interact with the database. The interactive part of the database is as follows:



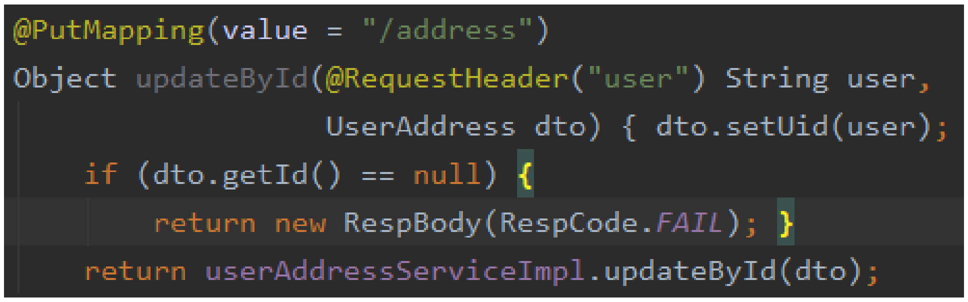
```
<insert id="insert" parameterType="UserAddress">
    INSERT INTO
        user_address
        (id, uid, name, phone, zip, note, main, stat, date,country,province,city,dist)
    VALUES
        (#{id}, #{uid}, #{name}, #{phone}, #{zip}, #{note}, #{main}, #{stat}, #{date},#{country},#{province},#{city},#{dist})
```

If the interaction with the database is successful, a message is returned to the insert() method of the corresponding UserAddressRepository:

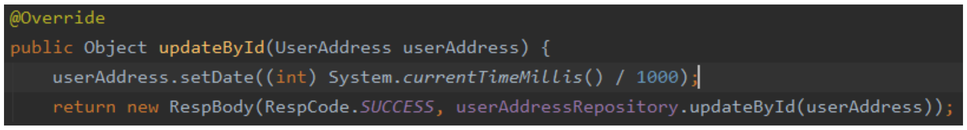

```
Object insert(UserAddress userAddress);
```

4. **updateById() Method:**

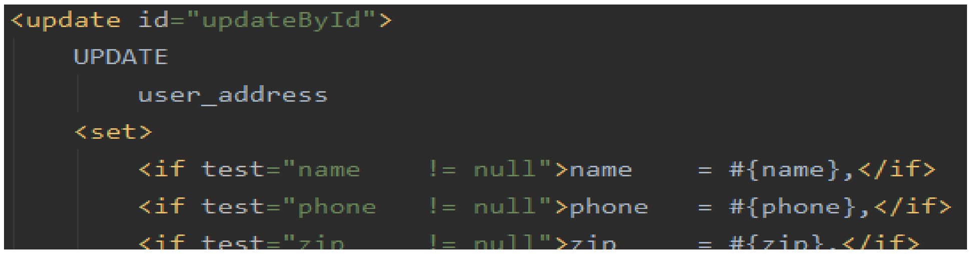
   updateById() uses the Put request method, and it is responsible for updating the user's address information by Id of this address and Id of this user.

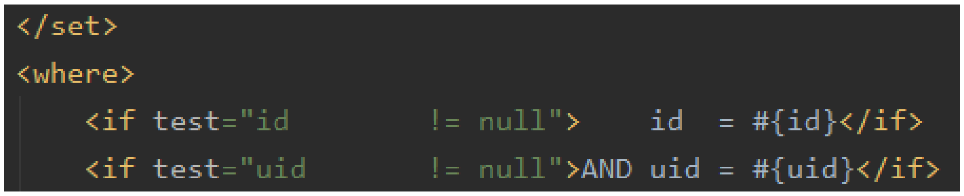

```
@PutMapping(value = "/address")
Object updateById(@RequestHeader("user") String user,
                  UserAddress dto) { dto.setUid(user);
    if (dto.getId() == null) {
        return new RespBody(RespCode.FAIL); }
    return userAddressServiceImpl.updateById(dto);
```

This updateById() method receives the put request to update the user address, and can carry parameters such as id, uid, name, phone, zip, note, main, stat, date, country, province, city, dist, as shown in the above figure of UserAddress class. And it calls the updateById() method of the implementation class userAddressServiceImpl of the interface userAddressService for further processing.

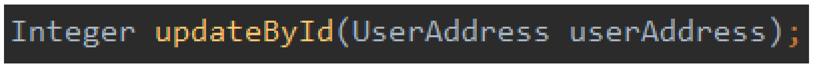

```
@Override
public Object updateById(UserAddress userAddress) {
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    return new RespBody(RespCode.SUCCESS, userAddressRepository.updateById(userAddress));
```

This updateById() method forwards the data to the dao layer to interact with the database. The interactive part of the database is as follows:


```
<update id="updateById">
    UPDATE
        user_address
    <set>
        <if test="name     != null">name     = #{name},</if>
        <if test="phone    != null">phone    = #{phone},</if>
        <if test="zip      != null">zip      = #{zip},</if>
```


```
</set>
<where>
    <if test="id          != null">    id  = #{id}</if>
    <if test="uid         != null">AND uid = #{uid}</if>
```

If the interaction with the database is successful, a message is returned to the updateById() method of the corresponding UserAddressRepository:

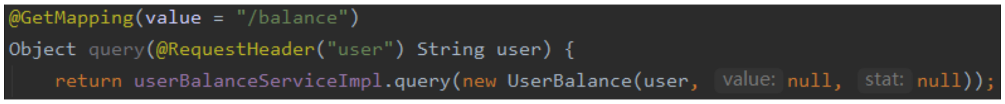

```
Integer updateById(UserAddress userAddress);
```

6.5 Management of User's Balance Information

UserBalanceController: This class is responsible for the logic of user balance information management: this class specifically performs operations on user balance. There are 3 methods in total, namely query, addValueByUId and subtractValueByUId.

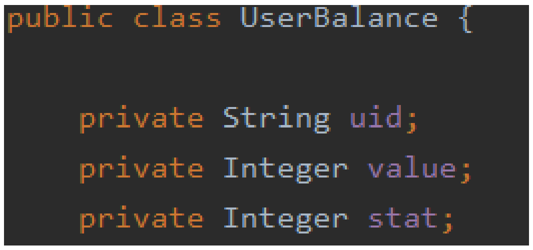
1. **query() Method:**

   query() uses the get request method, and it is responsible for querying the user's balance information.



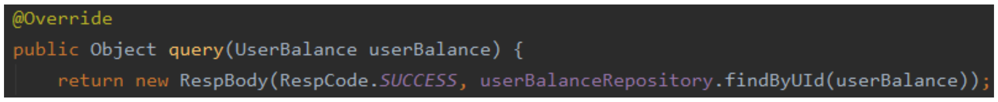
```
@GetMapping(value = "/balance")
Object query(@RequestHeader("user") String user) {
    return userBalanceServiceImpl.query(new UserBalance(user,  value: null,  stat: null));
```

This query() method receives the get request to query the user balance, and can carry parameters such as uid, value, stat, as shown in the following figure:



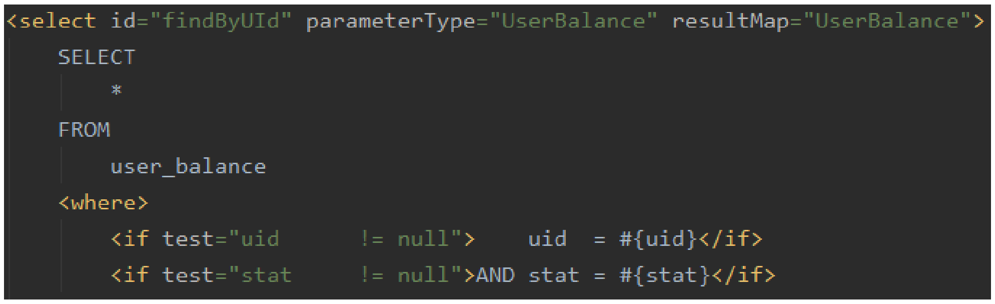
```
public class UserBalance {

    private String uid;
    private Integer value;
    private Integer stat;
```

And it calls the query() method of the implementation class userBalanceServiceImpl of the interface userBalanceService for further processing.



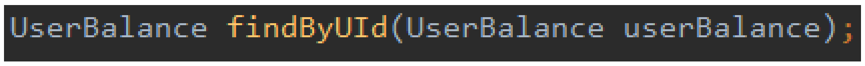
```
@Override
public Object query(UserBalance userBalance) {
    return new RespBody(RespCode.SUCCESS, userBalanceRepository.findByUId(userBalance));
```

This query() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



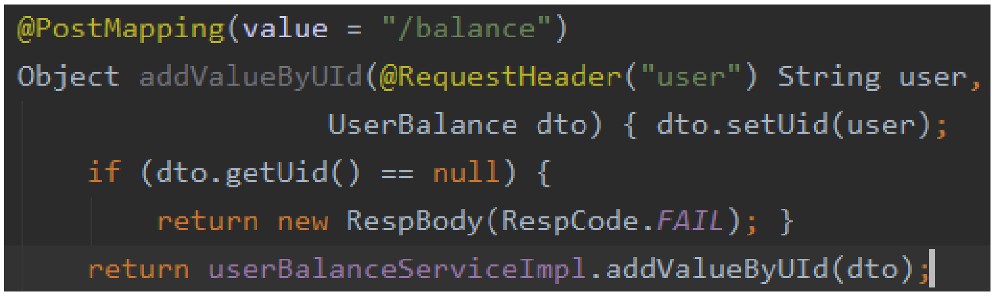
```
<select id="findByUId" parameterType="UserBalance" resultMap="UserBalance">
    SELECT
        *
    FROM
        user_balance
    <where>
        <if test="uid      != null">    uid  = #{uid}</if>
        <if test="stat     != null">AND stat = #{stat}</if>
```

If the interaction with the database is successful, the resultMap object is returned to the findByUId() method of the corresponding UserBalanceRepository:

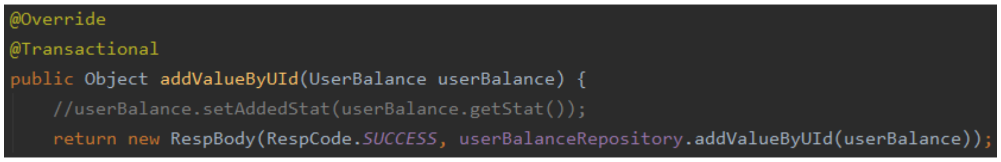

```
UserBalance findByUId(UserBalance userBalance);
```

2. **addValueByUId() Method:**

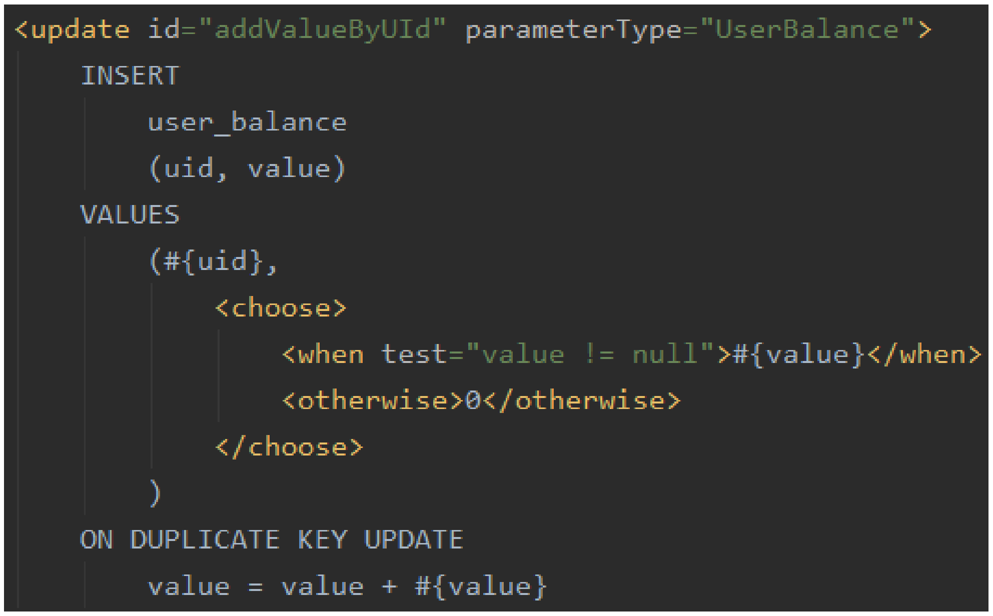
addValueByUId() uses the Post request method, and it is responsible for adding value in the user's balance.


```
@PostMapping(value = "/balance")
Object addValueByUId(@RequestHeader("user") String user,
                     UserBalance dto) { dto.setUid(user);
    if (dto.getUid() == null) {
        return new RespBody(RespCode.FAIL); }
    return userBalanceServiceImpl.addValueByUId(dto);
```

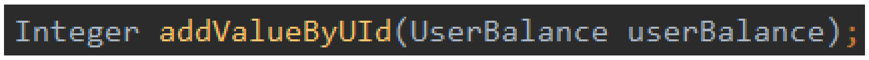
This addValueByUId() method receives the post request to add value in user's balance, and can carry parameters such as uid, value, stat, as shown in the above figure of UserBalance class. And it calls the addValueByUId() method of the implementation class userBalanceServiceImpl of the interface userBalanceService for further processing.


```
@Override
@Transactional
public Object addValueByUId(UserBalance userBalance) {
    //userBalance.setAddedStat(userBalance.getStat());
    return new RespBody(RespCode.SUCCESS, userBalanceRepository.addValueByUId(userBalance));
```

This addValueByUId() method forwards the data to the dao layer to interact with the database. The interactive part of the database is as follows:


```
<update id="addValueByUId" parameterType="UserBalance">
    INSERT
        user_balance
        (uid, value)
    VALUES
        (#{uid},
            <choose>
                <when test="value != null">#{value}</when>
                <otherwise>0</otherwise>
            </choose>
        )
    ON DUPLICATE KEY UPDATE
        value = value + #{value}
```
If the interaction with the database is successful, a message is returned to the

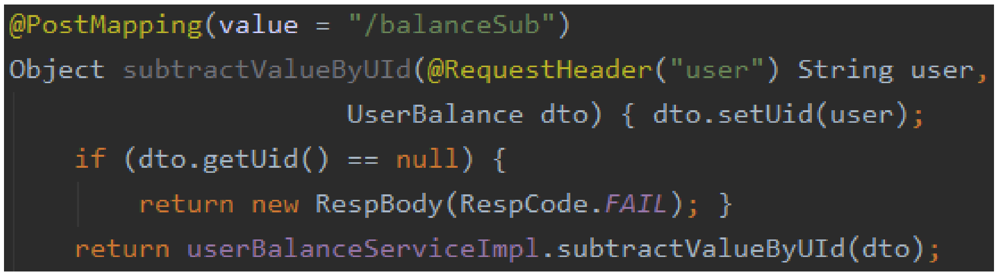
addValueByUId() method of the corresponding UserBalanceRepository:



```
Integer addValueByUId(UserBalance userBalance);
```

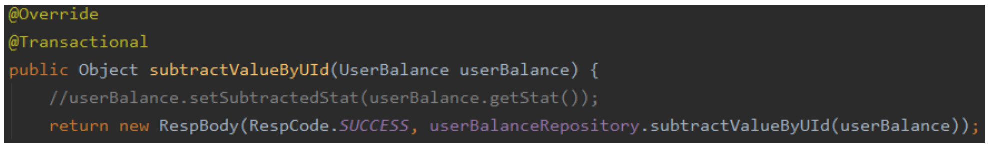
3. **subtractValueByUId() Method:**

subtractValueByUId() uses the Post request method, and it is responsible for subtracting value in the user's balance.



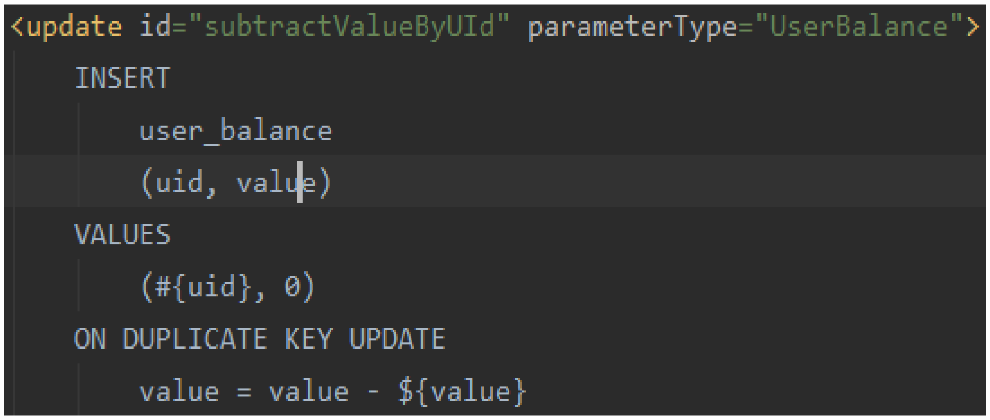
```
@PostMapping(value = "/balanceSub")
Object subtractValueByUId(@RequestHeader("user") String user,
                     UserBalance dto) { dto.setUid(user);
    if (dto.getUid() == null) {
        return new RespBody(RespCode.FAIL); }
    return userBalanceServiceImpl.subtractValueByUId(dto);
```

This subtractValueByUId() method receives the post request to subtract value in user’s balance, and can carry parameters such as uid, value, stat, as shown in the above figure of UserBalance class. And it calls the subtractValueByUId() method of the implementation class userBalanceServiceImpl of the interface userBalanceService for further processing.



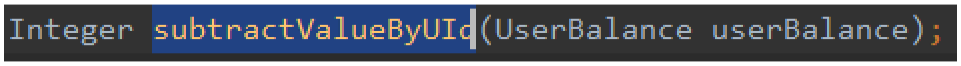
```
@Override
@Transactional
public Object subtractValueByUId(UserBalance userBalance) {
    //userBalance.setSubtractedStat(userBalance.getStat());
    return new RespBody(RespCode.SUCCESS, userBalanceRepository.subtractValueByUId(userBalance));
```

This subtractValueByUId() method forwards the data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<update id="subtractValueByUId" parameterType="UserBalance">
    INSERT
        user_balance
        (uid, value)
    VALUES
        (#{uid}, 0)
    ON DUPLICATE KEY UPDATE
        value = value - ${value}
```

If the interaction with the database is successful, a message is returned to the subtractValueByUId() method of the corresponding UserBalanceRepository:



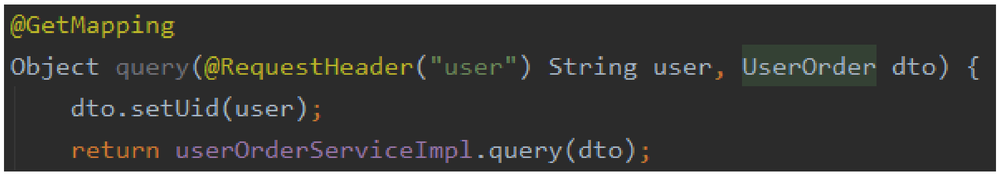
```
Integer subtractValueByUId(UserBalance userBalance);
```

6.6 Management of User's Order Information

UserOrderController: This class is responsible for the logic of user order information management: this class specifically performs operations on user order. There are 2 methods in total, namely query and insert.

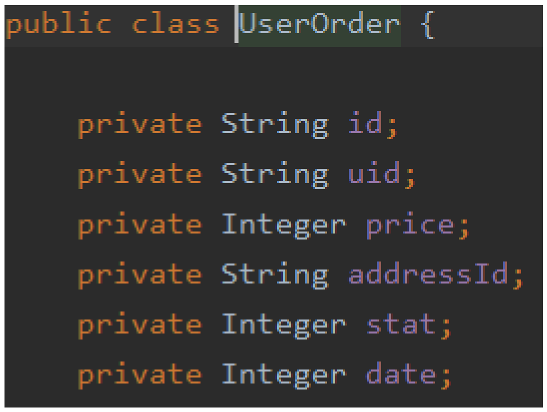
1. **query() Method:**

   query() uses the get request method, and it is responsible for querying the user's order information.



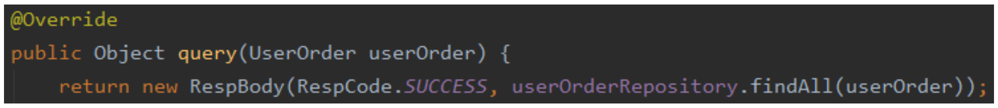
```
@GetMapping
Object query(@RequestHeader("user") String user, UserOrder dto) {
    dto.setUid(user);
    return userOrderServiceImpl.query(dto);
```

This query() method receives the get request to query the user order, and can carry parameters such as id, uid, price, addressId, stat, date, as shown in the following figure:



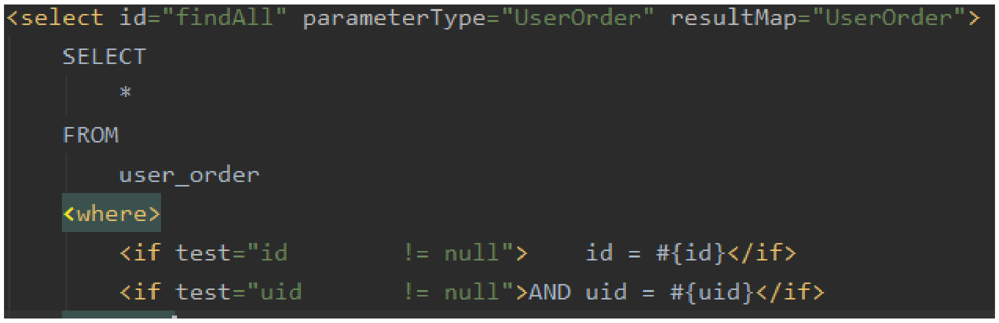
```
public class UserOrder {

    private String id;
    private String uid;
    private Integer price;
    private String addressId;
    private Integer stat;
    private Integer date;
```

And it calls the query() method of the implementation class userOrderServiceImpl of the interface userOrderService for further processing.



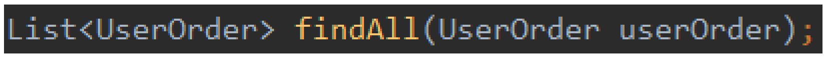
```
@Override
public Object query(UserOrder userOrder) {
    return new RespBody(RespCode.SUCCESS, userOrderRepository.findAll(userOrder));
```

This query() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<select id="findAll" parameterType="UserOrder" resultMap="UserOrder">
    SELECT
        *
    FROM
        user_order
    <where>
        <if test="id          != null">     id = #{id}</if>
        <if test="uid         != null">AND uid = #{uid}</if>
```

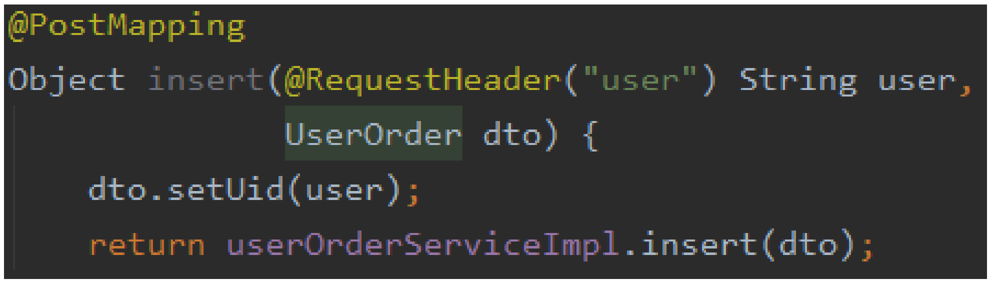
If the interaction with the database is successful, the resultMap object is returned to the findAll() method of the corresponding UserOrderRepository:



```
List<UserOrder> findAll(UserOrder userOrder);
```

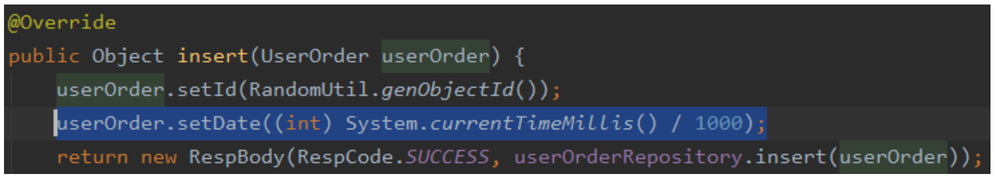
2. **insert() Method:**

insert() uses the Post request method, and it is responsible for adding new order in the user's order information.



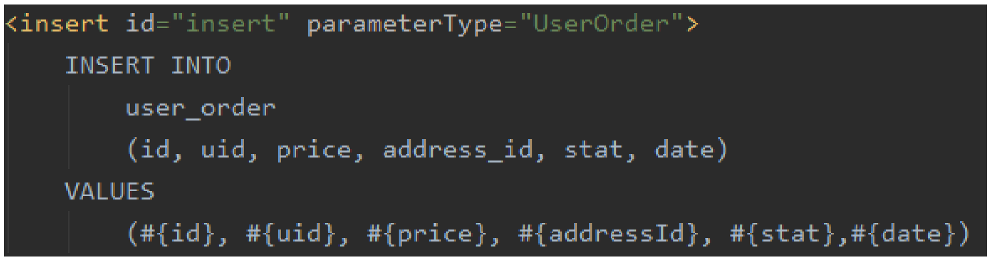
```
@PostMapping
Object insert(@RequestHeader("user") String user,
              UserOrder dto) {
    dto.setUid(user);
    return userOrderServiceImpl.insert(dto);
```

This insert() method receives the post request to add order in user’s order information, and can carry parameters such as uid, value, stat, as shown in the above figure of UserOrder class. And it calls the insert() method of the implementation class userOrderServiceImpl of the interface userOrderService for further processing.



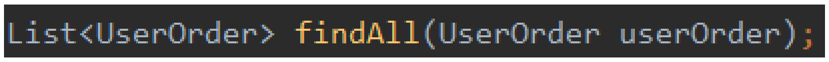
```
@Override
public Object insert(UserOrder userOrder) {
    userOrder.setId(RandomUtil.genObjectId());
    userOrder.setDate((int) System.currentTimeMillis() / 1000);
    return new RespBody(RespCode.SUCCESS, userOrderRepository.insert(userOrder));
```

This insert() method forwards the data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<insert id="insert" parameterType="UserOrder">
    INSERT INTO
        user_order
        (id, uid, price, address_id, stat, date)
    VALUES
        (#{id}, #{uid}, #{price}, #{addressId}, #{stat},#{date})
```

If the interaction with the database is successful, a message is returned to the insert() method of the corresponding UserOrderRepository:



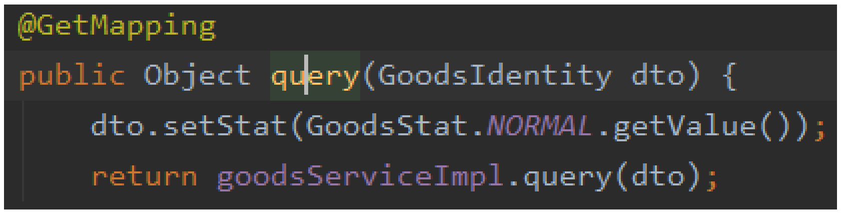
```
List<UserOrder> findAll(UserOrder userOrder);
```

6.7 Management of Goods Information

GoodsController: This class is responsible for the logic of goods information management: this class specifically performs operations on goods. There are 3 methods in total, namely query, insert and updateById.

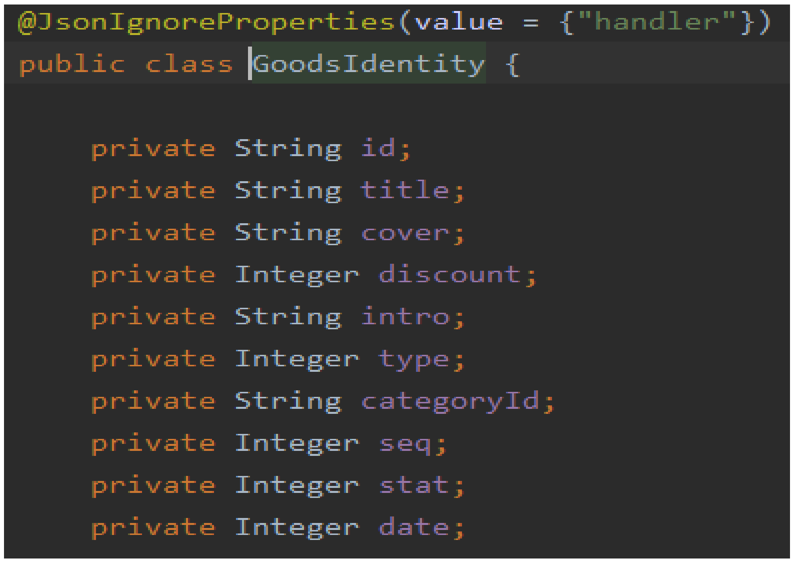
1. query() Method:

query() uses the get request method, and it is responsible for querying the whole goods information by implicates parts informatio of this goods.



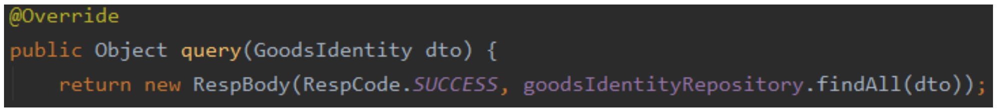
```
@GetMapping
public Object query(GoodsIdentity dto) {
    dto.setStat(GoodsStat.NORMAL.getValue());
    return goodsServiceImpl.query(dto);
```

This query() method receives the get request to query the goods, and can carry parameters such as id, title, type, stat, date, as shown in the following figure:



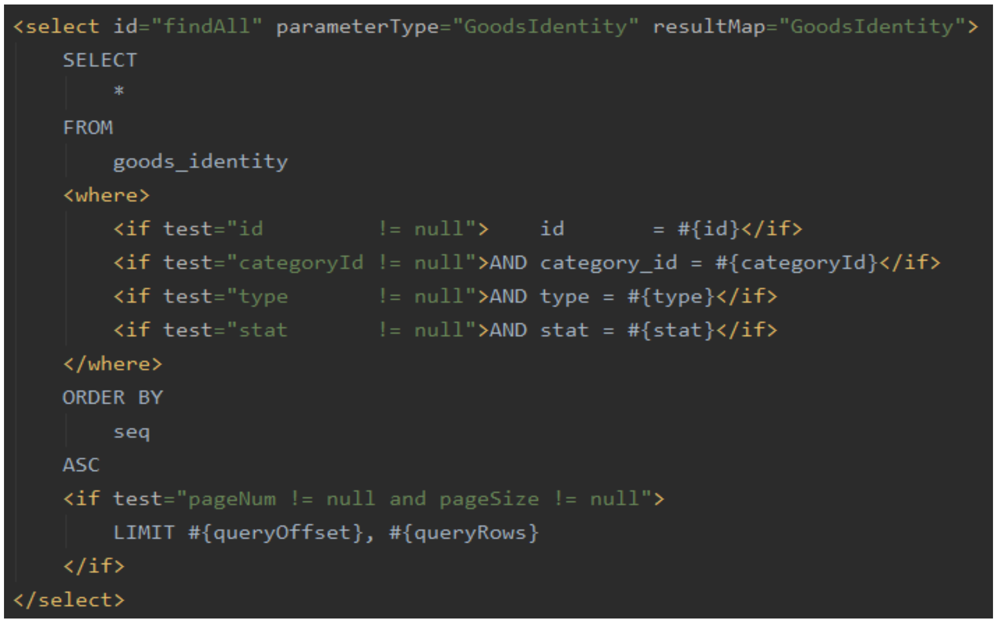
```
@JsonIgnoreProperties(value = {"handler"})
public class GoodsIdentity {

    private String id;
    private String title;
    private String cover;
    private Integer discount;
    private String intro;
    private Integer type;
    private String categoryId;
    private Integer seq;
    private Integer stat;
    private Integer date;
```

And it calls the query() method of the implementation class GoodsServiceImpl of the interface GoodsService for further processing.



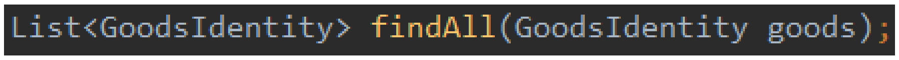
```
@Override
public Object query(GoodsIdentity dto) {
    return new RespBody(RespCode.SUCCESS, goodsIdentityRepository.findAll(dto));
```

This query() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<select id="findAll" parameterType="GoodsIdentity" resultMap="GoodsIdentity">
    SELECT
        *
    FROM
        goods_identity
    <where>
        <if test="id         != null">    id         = #{id}</if>
        <if test="categoryId != null">AND category_id = #{categoryId}</if>
        <if test="type       != null">AND type = #{type}</if>
        <if test="stat       != null">AND stat = #{stat}</if>
    </where>
    ORDER BY
        seq
    ASC
    <if test="pageNum != null and pageSize != null">
        LIMIT #{queryOffset}, #{queryRows}
    </if>
</select>
```

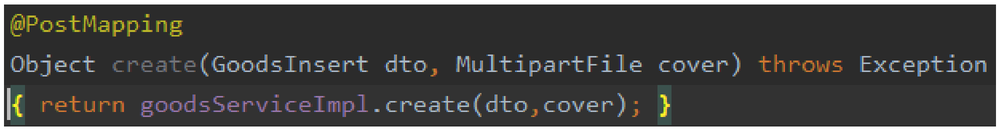
If the interaction with the database is successful, the resultMap object will be returned to the related findAll() method of the GoodsIdentityRepository.java:



```
List<GoodsIdentity> findAll(GoodsIdentity goods);
```

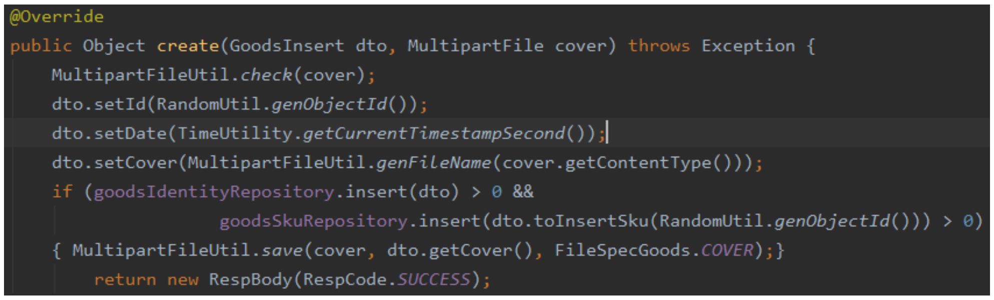
2. **create() Method:**

create() uses the Post request method, and it is responsible for ceating and adding new goods in the database.



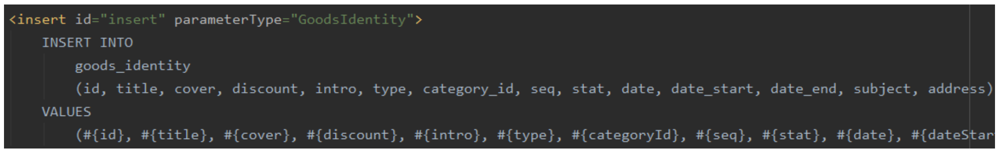
```
@PostMapping
Object create(GoodsInsert dto, MultipartFile cover) throws Exception
{ return goodsServiceImpl.create(dto,cover); }
```

This create() method receives the post request to add goods in database, and can carry parameters such as id, title, type, stat, date, as shown in the above figure of GoodsIdentity class. And it calls the create() method of the implementation class GoodsServiceImpl of the interface GoodsService for further processing.

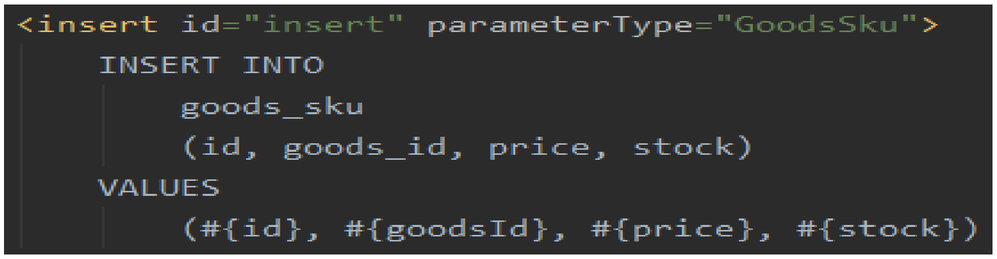


```
@Override
public Object create(GoodsInsert dto, MultipartFile cover) throws Exception {
    MultipartFileUtil.check(cover);
    dto.setId(RandomUtil.genObjectId());
    dto.setDate(TimeUtility.getCurrentTimestampSecond());
    dto.setCover(MultipartFileUtil.genFileName(cover.getContentType()));
    if (goodsIdentityRepository.insert(dto) > 0 &&
                    goodsSkuRepository.insert(dto.toInsertSku(RandomUtil.genObjectId())) > 0)
    { MultipartFileUtil.save(cover, dto.getCover(), FileSpecGoods.COVER);}
        return new RespBody(RespCode.SUCCESS);
```

This create() method forwards the data to the dao layer to interact with the database. This method will do 2 different interactions with database. goodsIdentityRepository.insert() method could add goods' indentity information into database. goodsSkuRepository.insert() method is responsible for add attribute properties, such as color, size. The 2 interactive parts of the database is as follows:

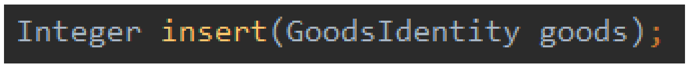


```
<insert id="insert" parameterType="GoodsIdentity">
    INSERT INTO
        goods_identity
        (id, title, cover, discount, intro, type, category_id, seq, stat, date, date_start, date_end, subject, address)
    VALUES
        (#{id}, #{title}, #{cover}, #{discount}, #{intro}, #{type}, #{categoryId}, #{seq}, #{stat}, #{date}, #{dateStar
```

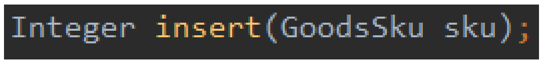


```
<insert id="insert" parameterType="GoodsSku">
    INSERT INTO
        goods_sku
        (id, goods_id, price, stock)
    VALUES
        (#{id}, #{goodsId}, #{price}, #{stock})
```

If the interactions with the database are successful, messages are separately returned to the insert() methods of the corresponding GoodsIdentityRepository and GoodsSkuRepository:



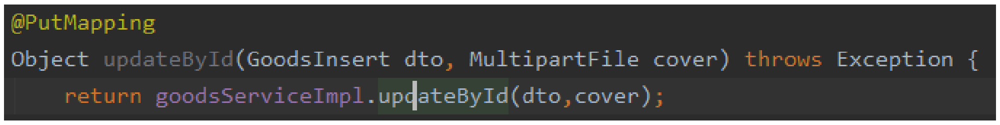
```
Integer insert(GoodsIdentity goods);
```



```
Integer insert(GoodsSku sku);
```

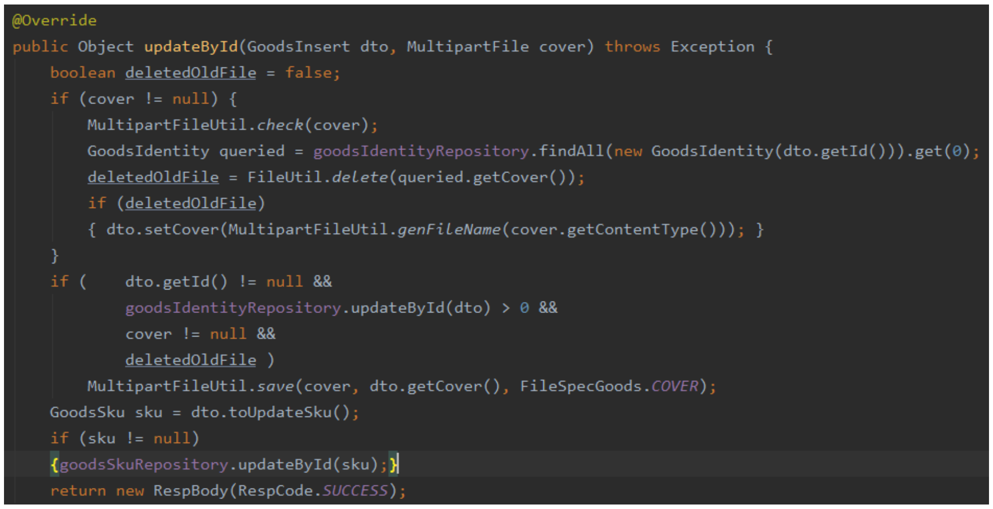
3. **updateById() Method:**

updateById () uses the put request method, and it is responsible for updating goods information by Id of this goods and its' cover file.



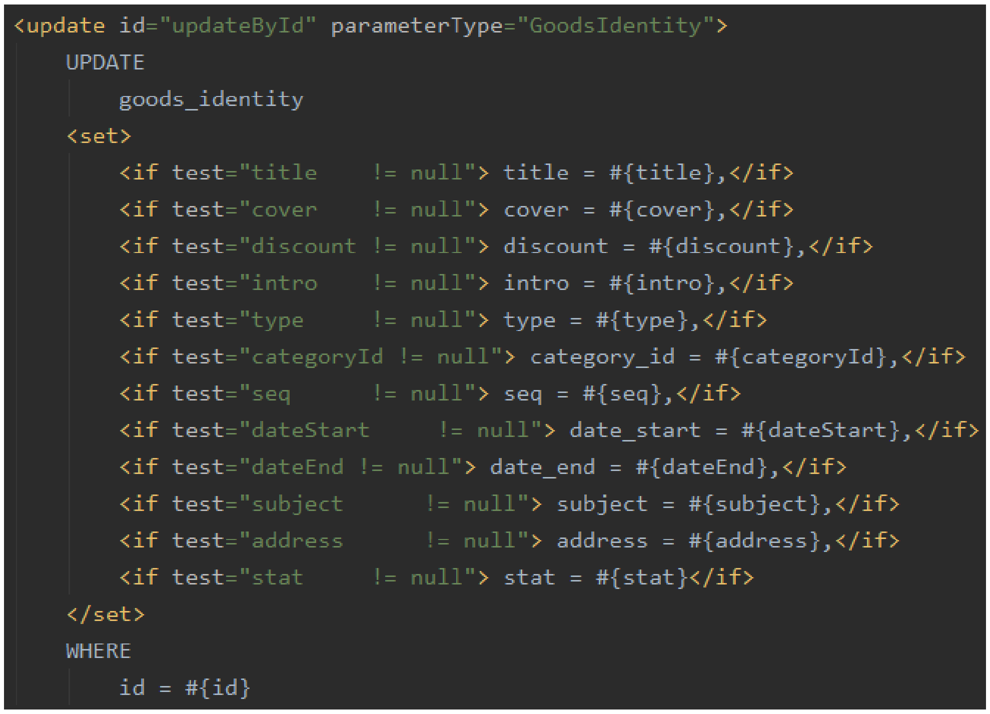
```
@PutMapping
Object updateById(GoodsInsert dto, MultipartFile cover) throws Exception {
    return goodsServiceImpl.updateById(dto,cover);
```

This updateById() method receives the put request to update the information of the goods, and can carry parameters such as id, title, type, stat, date and cover file, as shown in the above figure of GoodsIdentity class. And it calls the updateById() method of the implementation class GoodsServiceImpl of the interface GoodsService for further processing.

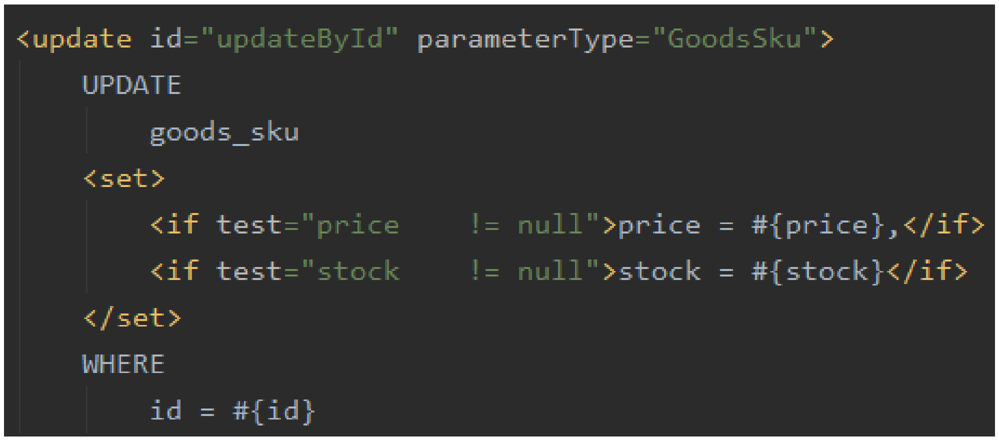


```java
@Override
public Object updateById(GoodsInsert dto, MultipartFile cover) throws Exception {
    boolean deletedOldFile = false;
    if (cover != null) {
        MultipartFileUtil.check(cover);
        GoodsIdentity queried = goodsIdentityRepository.findAll(new GoodsIdentity(dto.getId())).get(0);
        deletedOldFile = FileUtil.delete(queried.getCover());
        if (deletedOldFile)
        { dto.setCover(MultipartFileUtil.genFileName(cover.getContentType())); }
    }
    if (    dto.getId() != null &&
            goodsIdentityRepository.updateById(dto) > 0 &&
            cover != null &&
            deletedOldFile )
        MultipartFileUtil.save(cover, dto.getCover(), FileSpecGoods.COVER);
    GoodsSku sku = dto.toUpdateSku();
    if (sku != null)
    {goodsSkuRepository.updateById(sku);}
    return new RespBody(RespCode.SUCCESS);
```

This updateById() method forwards the data to the dao layer to interact with the database. This method will do 2 different interactions with database. goodsIdentityRepository.updateById() method could to update goods' indentity information in database. goodsSkuRepository.updateById() method is responsible for updating attribute properties, such as color, size. The 2 interactive parts of the database is as follows:

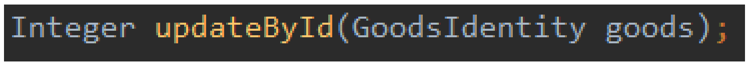


```xml
<update id="updateById" parameterType="GoodsIdentity">
    UPDATE
        goods_identity
    <set>
        <if test="title    != null"> title = #{title},</if>
        <if test="cover    != null"> cover = #{cover},</if>
        <if test="discount != null"> discount = #{discount},</if>
        <if test="intro    != null"> intro = #{intro},</if>
        <if test="type     != null"> type = #{type},</if>
        <if test="categoryId != null"> category_id = #{categoryId},</if>
        <if test="seq      != null"> seq = #{seq},</if>
        <if test="dateStart     != null"> date_start = #{dateStart},</if>
        <if test="dateEnd != null"> date_end = #{dateEnd},</if>
        <if test="subject      != null"> subject = #{subject},</if>
        <if test="address      != null"> address = #{address},</if>
        <if test="stat     != null"> stat = #{stat}</if>
    </set>
    WHERE
        id = #{id}
```

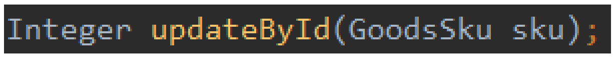


```
<update id="updateById" parameterType="GoodsSku">
    UPDATE
        goods_sku
    <set>
        <if test="price    != null">price = #{price},</if>
        <if test="stock    != null">stock = #{stock}</if>
    </set>
    WHERE
        id = #{id}
```

If the interactions with the database are successful, messages are separately returned to the updateById() method of the corresponding GoodsIdentityRepository and GoodsSkuRepository:



```
Integer updateById(GoodsIdentity goods);
```



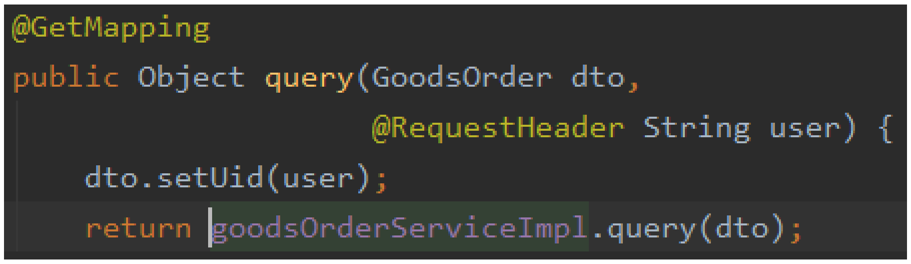
```
Integer updateById(GoodsSku sku);
```

6.8 Management of Order’s Goods Information

GoodsOrderController: This class is responsible for the logic of order and its’ goods information management: this class specifically performs operations on order. There are 3 methods at present stage, namely query, genOrder and updateById.

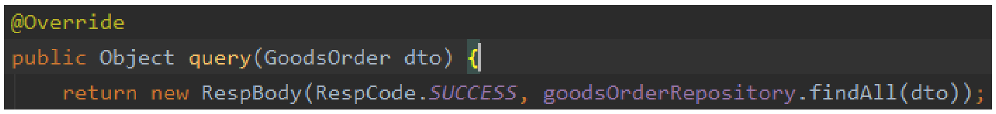
1. **query() Method:**

   query() uses the get request method, and it is responsible for querying the order information by user Id.



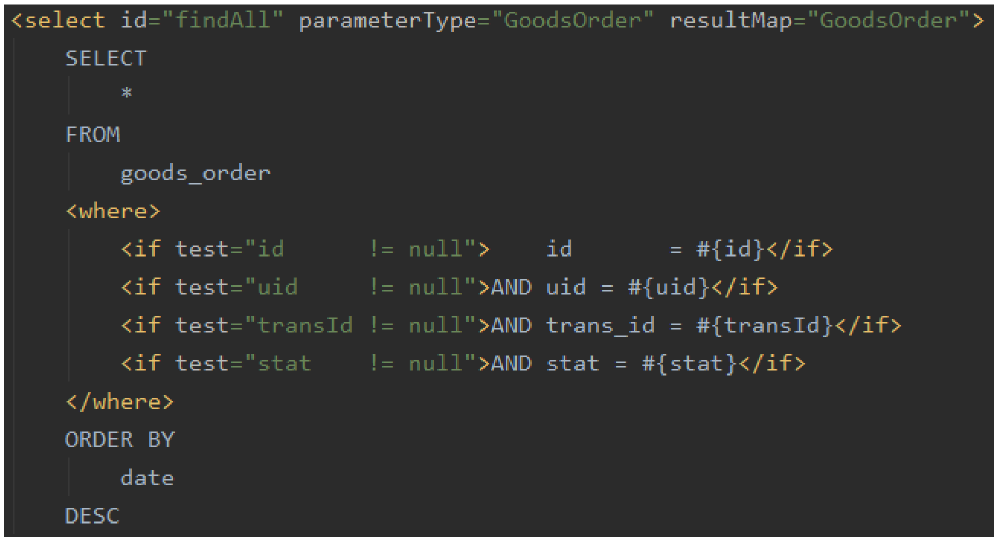
```
@GetMapping
public Object query(GoodsOrder dto,
                    @RequestHeader String user) {
    dto.setUid(user);
    return goodsOrderServiceImpl.query(dto);
```

This query() method receives the get request to query the order information by order’s user Id. And it calls the query() method of the implementation class GoodsOrderServiceImpl of the interface GoodsOrderService for further processing.



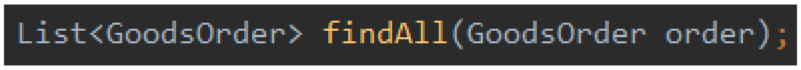
```
@Override
public Object query(GoodsOrder dto) {
    return new RespBody(RespCode.SUCCESS, goodsOrderRepository.findAll(dto));
```

This query() method forwards the request data to the dao layer to interact with the database. The interactive part of the database is as follows:



```
<select id="findAll" parameterType="GoodsOrder" resultMap="GoodsOrder">
    SELECT
        *
    FROM
        goods_order
    <where>
        <if test="id      != null">    id       = #{id}</if>
        <if test="uid     != null">AND uid = #{uid}</if>
        <if test="transId != null">AND trans_id = #{transId}</if>
        <if test="stat    != null">AND stat = #{stat}</if>
    </where>
    ORDER BY
        date
    DESC
```

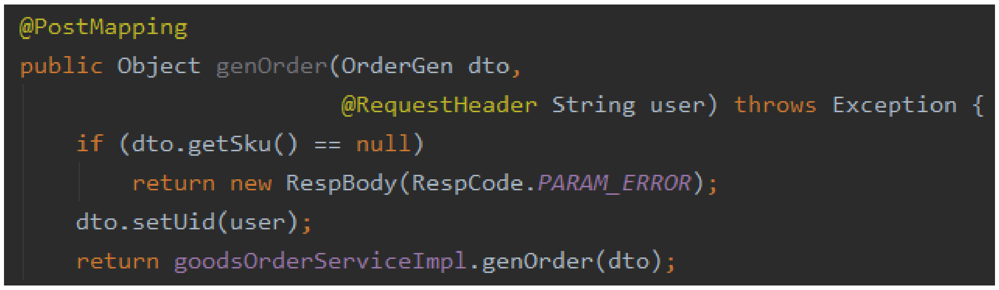
If the interaction with the database is successful, the resultMap object is returned to the findAll() method of the corresponding GoodsOrderRepository:



```
List<GoodsOrder> findAll(GoodsOrder order);
```

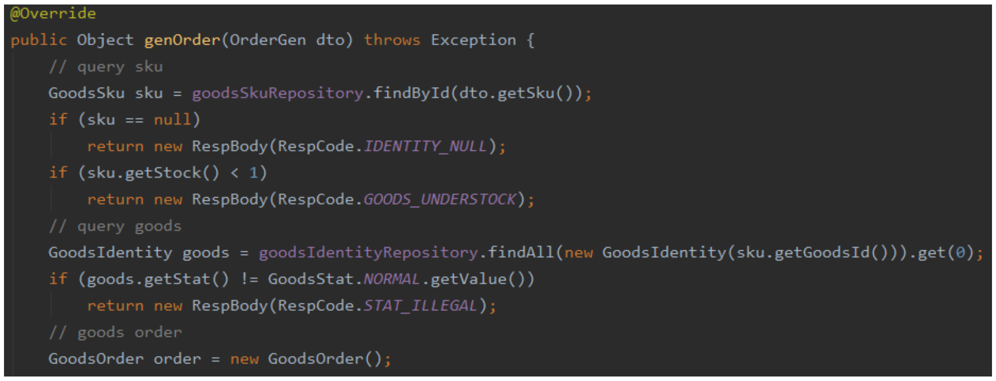
2. genOrder() Method:

genOrder() uses the Post request method, and it is responsible for creating new order which associate with user Id.

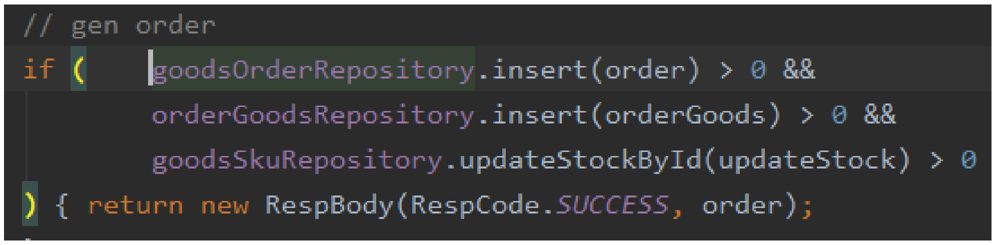


```
@PostMapping
public Object genOrder(OrderGen dto,
                       @RequestHeader String user) throws Exception {
    if (dto.getSku() == null)
        return new RespBody(RespCode.PARAM_ERROR);
    dto.setUid(user);
    return goodsOrderServiceImpl.genOrder(dto);
```

This genOrder() method receives the post request to generate new order, and can carry parameters such as uid, sku and comment. And it calls the genOrder() method of the implementation class GoodsOrderServiceImpl of the interface GoodsOrderService for further processing.

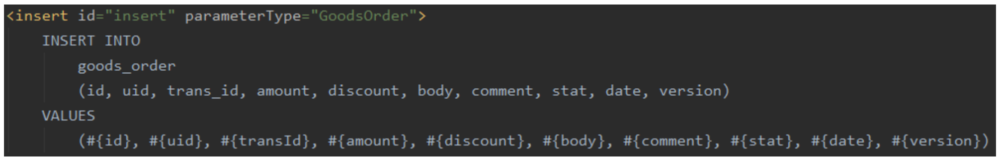


```
@Override
public Object genOrder(OrderGen dto) throws Exception {
    // query sku
    GoodsSku sku = goodsSkuRepository.findById(dto.getSku());
    if (sku == null)
        return new RespBody(RespCode.IDENTITY_NULL);
    if (sku.getStock() < 1)
        return new RespBody(RespCode.GOODS_UNDERSTOCK);
    // query goods
    GoodsIdentity goods = goodsIdentityRepository.findAll(new GoodsIdentity(sku.getGoodsId())).get(0);
    if (goods.getStat() != GoodsStat.NORMAL.getValue())
        return new RespBody(RespCode.STAT_ILLEGAL);
    // goods order
    GoodsOrder order = new GoodsOrder();
```

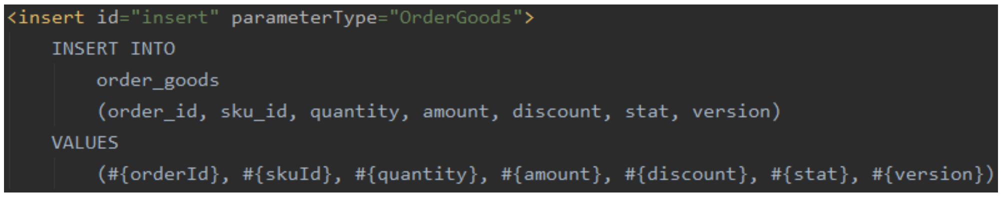


```
// gen order
if (     goodsOrderRepository.insert(order) > 0 &&
         orderGoodsRepository.insert(orderGoods) > 0 &&
         goodsSkuRepository.updateStockById(updateStock) > 0
) { return new RespBody(RespCode.SUCCESS, order);
```

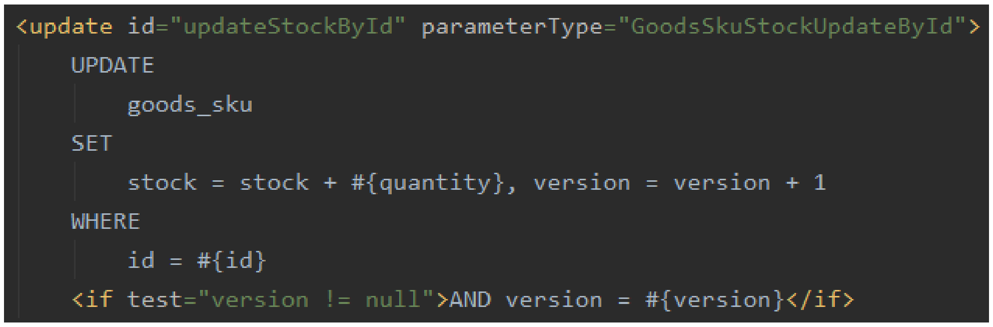
This genOrder() method forwards the data to the dao layer to interact with the database. This method will do 3 different interactions with database. GoodsOrderRepository.insert() method could to add new type of order which is goods' Order in database. OrderGoodsRepository.insert() method could to add information of all goods of this order in the database. goodsSkuRepository.updateStockById() method is responsible for updating stock information of these goods after the generation of this order. The 3 interactive parts of the database is as follows:



```
<insert id="insert" parameterType="GoodsOrder">
    INSERT INTO
        goods_order
        (id, uid, trans_id, amount, discount, body, comment, stat, date, version)
    VALUES
        (#{id}, #{uid}, #{transId}, #{amount}, #{discount}, #{body}, #{comment}, #{stat}, #{date}, #{version})
```

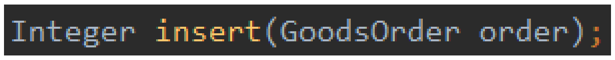


```
<insert id="insert" parameterType="OrderGoods">
    INSERT INTO
        order_goods
        (order_id, sku_id, quantity, amount, discount, stat, version)
    VALUES
        (#{orderId}, #{skuId}, #{quantity}, #{amount}, #{discount}, #{stat}, #{version})
```

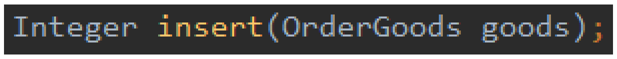


```
<update id="updateStockById" parameterType="GoodsSkuStockUpdateById">
    UPDATE
        goods_sku
    SET
        stock = stock + #{quantity}, version = version + 1
    WHERE
        id = #{id}
    <if test="version != null">AND version = #{version}</if>
```

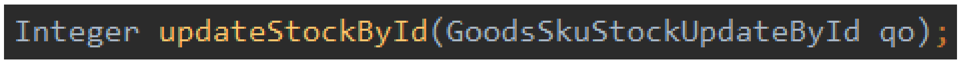
If the interactions with the database are successful, messages are separately returned to the insert() and updateStockById() method of the corresponding GoodsOrderRepository, OrderGoodsRepository and GoodsSkuRepository:



```
Integer insert(GoodsOrder order);
```



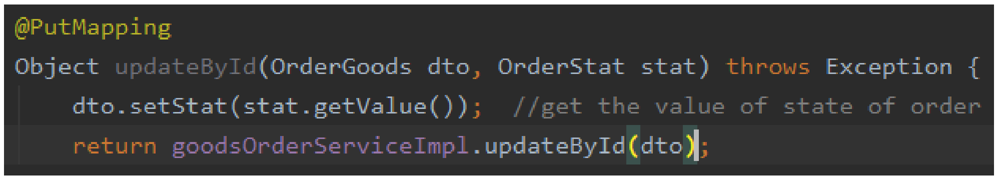
```
Integer insert(OrderGoods goods);
```



```
Integer updateStockById(GoodsSkuStockUpdateById qo);
```

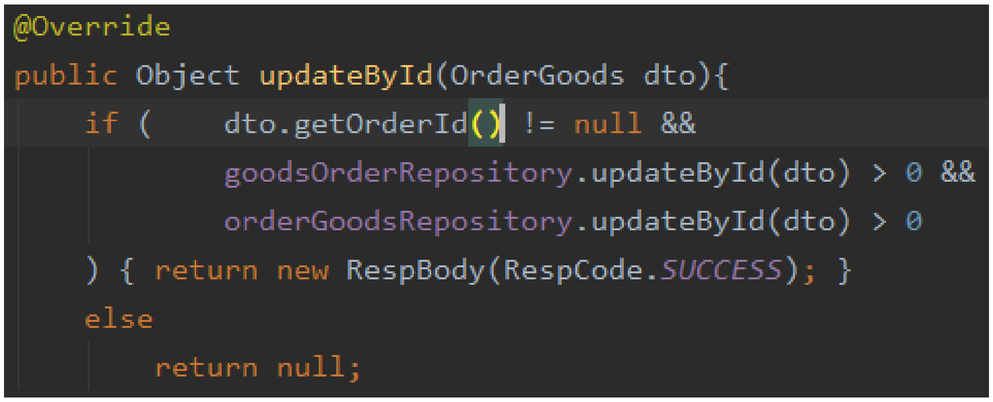
3. **updateById() Method:**

updateById () uses the put request method, and it is responsible for updating the state of order by Id of this order.



```
@PutMapping
Object updateById(OrderGoods dto, OrderStat stat) throws Exception {
    dto.setStat(stat.getValue());  //get the value of state of order
    return goodsOrderServiceImpl.updateById(dto);
```

This updateById() method receives the put request to update the state of the order, and can carry parameters such as id and stat. And it calls the updateById() method of the implementation class GoodsOrderServiceImpl of the interface GoodsOrderService for further processing.



```
@Override
public Object updateById(OrderGoods dto){
    if (     dto.getOrderId() != null &&
             goodsOrderRepository.updateById(dto) > 0 &&
             orderGoodsRepository.updateById(dto) > 0
    ) { return new RespBody(RespCode.SUCCESS); }
    else
        return null;
```

This updateById() method forwards the data to the dao layer to interact with the database. This method will do 2 different interactions with database. goodsOrderRepository.updateById() method could to update order's state in database. orderGoodsRepository.updateById() method is responsible for updating same order's state in database. The 2 interactive parts of the database is as follows:

```
<update id="updateById" parameterType="GoodsOrder">
    UPDATE
        goods_order
    <set>
        <if test="transId  != null">trans_id = #{transId},</if>
        <if test="stat     != null">stat = #{stat}</if>
    </set>
    WHERE
        id = #{id}
```

```
<update id="updateById" parameterType="OrderGoods">
    UPDATE
        order_goods
    <set>
        <if test="stat     != null">stat = #{stat}</if>
    </set>
    WHERE
        id = #{id}
```

If the interactions with the database are successful, messages are separately returned to the updateById() method of the corresponding GoodsOrderRepository and OrderGoodsRepository:

```
Integer updateById(OrderGoods order);
```

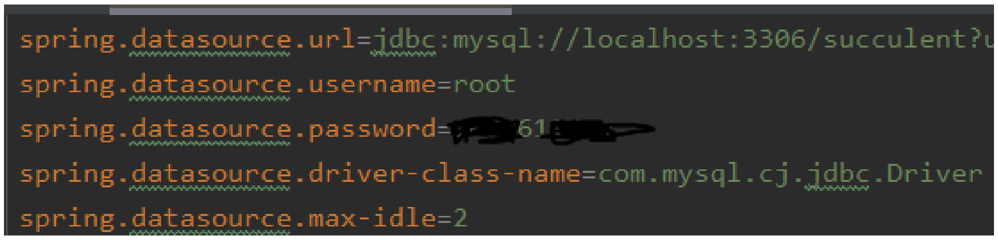
```
Integer updateById(OrderGoods goods);
```

6.9 Connection with MySQL Database

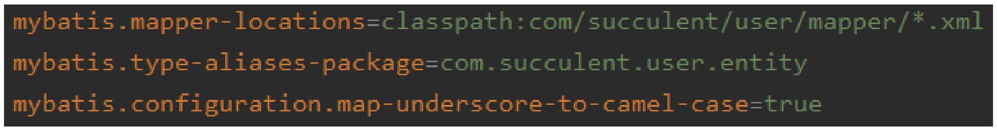
Install MySQL software on the computer. Add modules for supporting MySQL. Add the corresponding dependency in the pom.xml file.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
```

Modify the application.properties configuration file and add the following code:



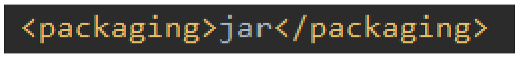
```
spring.datasource.url=jdbc:mysql://localhost:3306/succulent?u
spring.datasource.username=root
spring.datasource.password=[redacted]
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.max-idle=2
```



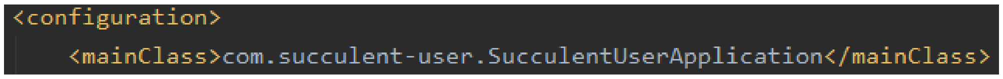
```
mybatis.mapper-locations=classpath:com/succulent/user/mapper/*.xml
mybatis.type-aliases-package=com.succulent.user.entity
mybatis.configuration.map-underscore-to-camel-case=true
```

6.10 Package Back End Part as Jar File

Set the startup class of the springboot project and add the following to pom.xml:

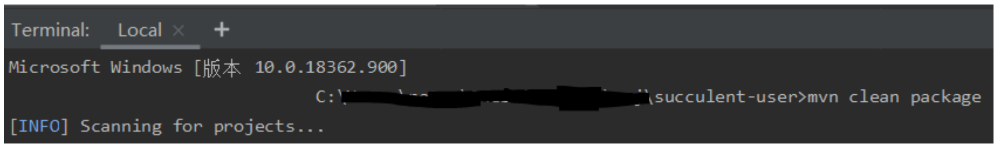


```
<packaging>jar</packaging>
```

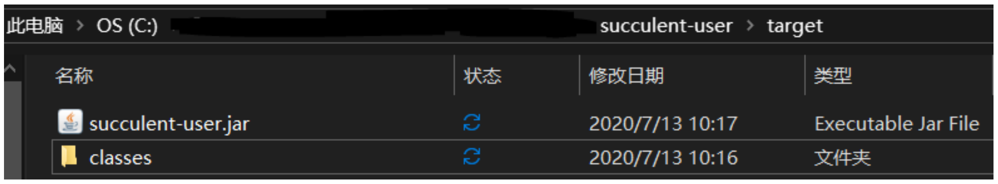


```
<configuration>
    <mainClass>com.succulent-user.SucculentUserApplication</mainClass>
```

Use Command Prompt or Intellij Terminal to enter the directory where the pom.xml of the springboot project is located. Run the command "mvn clean package" and the project will be packaged into the target folder.



```
Terminal:  Local ×  +
Microsoft Windows [版本 10.0.18362.900]
                         C:[redacted]\succulent-user>mvn clean package
[INFO] Scanning for projects...
```



此电脑 › OS (C:) [redacted] succulent-user › target

名称	状态	修改日期	类型
succulent-user.jar		2020/7/13 10:17	Executable Jar File
classes		2020/7/13 10:16	文件夹

7 Test of the Result

Because when I left the company, the project was not completed, so I could not test my works with the connection of Front End of this project. So here, I am going to show the steps of test of each funtions and its result.

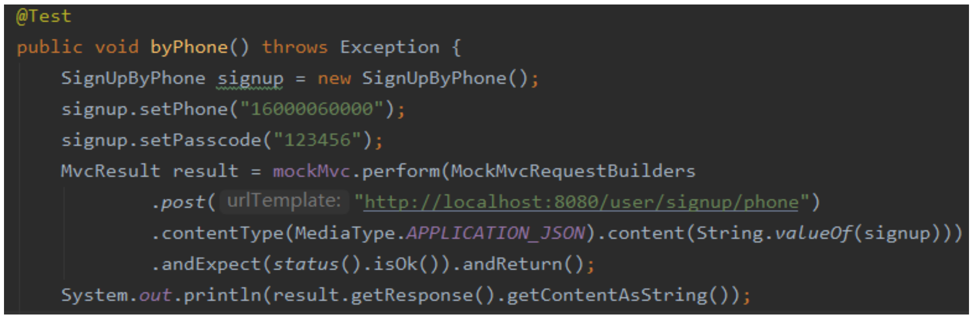
I built the project running environment on my personal computer. By creating Unit test with MockMvc tool for each Controller class and Service class, I tested functions that I have completed so far. The following shows the steps of test and its reslut:

7.1 Test of Controller (Integration Test)

Test the integration and calling relationships between various modules. Find out the program structure, module calling relationship and interface between modules related to software design.

1. Test of UserSignUpController:

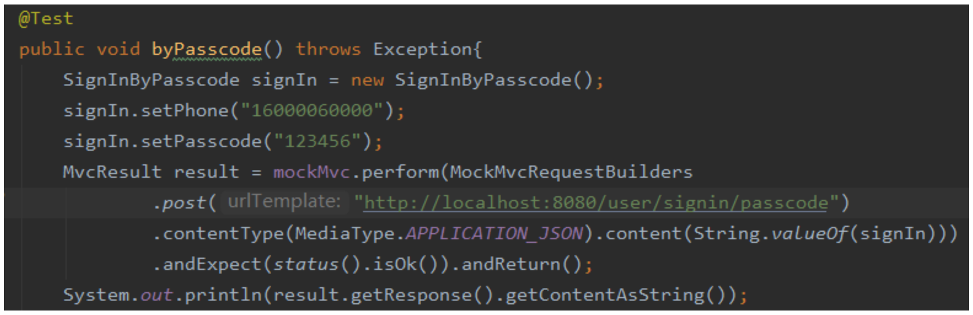
   Use mockMvc.perform() to execute the post request. Use MockMvcRequestBuilders.post("http://localhost:8080/user/signup/phone") to construct this request. .header() and .content() for passing parameters. .andExpect() adds an assertion after execution is complete. .andReturn() means to return the corresponding result after the execution is completed.



```
@Test
public void byPhone() throws Exception {
    SignUpByPhone signup = new SignUpByPhone();
    signup.setPhone("16000060000");
    signup.setPasscode("123456");
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/signup/phone")
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(signup)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

2. Test of UserSignInController:

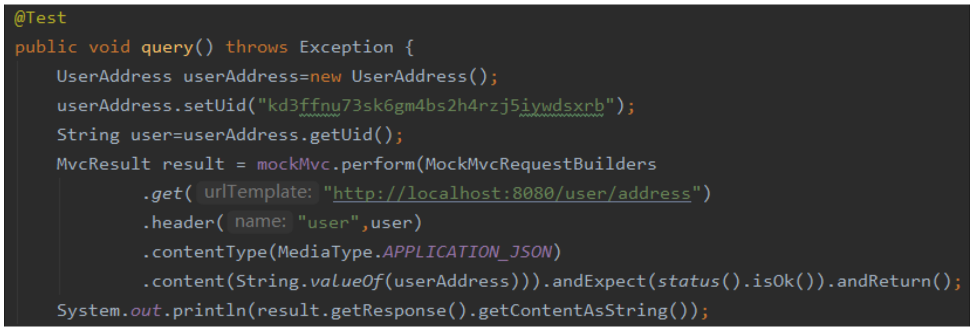
   Use mockMvc.perform() to execute the post request. Use MockMvcRequestBuilders.post("http://localhost:8080/user/signin/passcode") to construct this request. .header() and .content() for passing parameters.



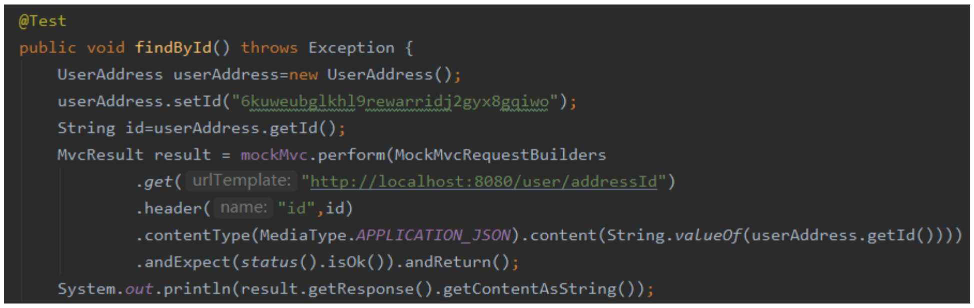
```
@Test
public void byPasscode() throws Exception{
    SignInByPasscode signIn = new SignInByPasscode();
    signIn.setPhone("16000060000");
    signIn.setPasscode("123456");
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/signin/passcode")
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(signIn)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

3. Test of UserAddressController:

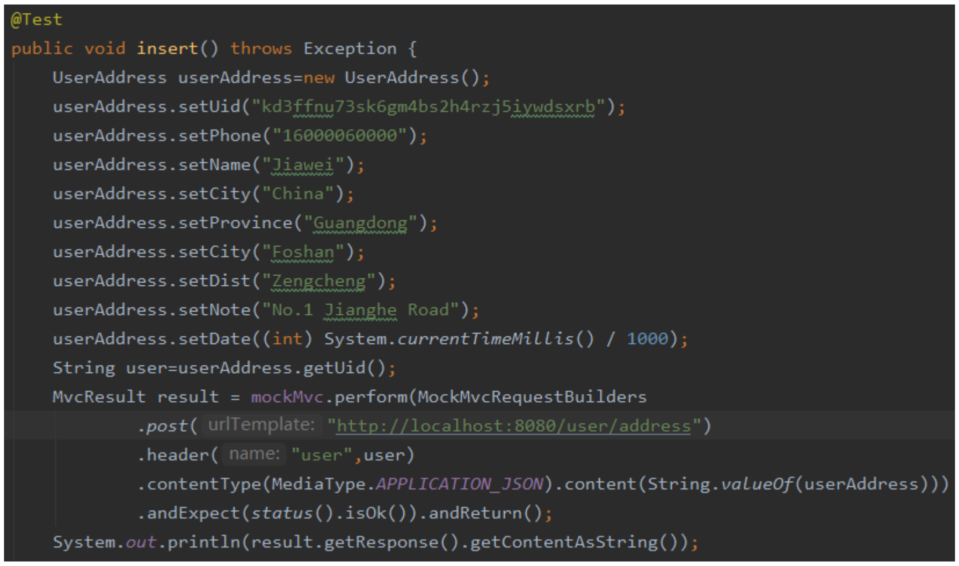
   Use mockMvc.perform() to execute the get/get/post/put request. Use MockMvcRequestBuilders.post("http://localhost:8080/user/address") to construct these requests. .header() and .content() for passing parameters.



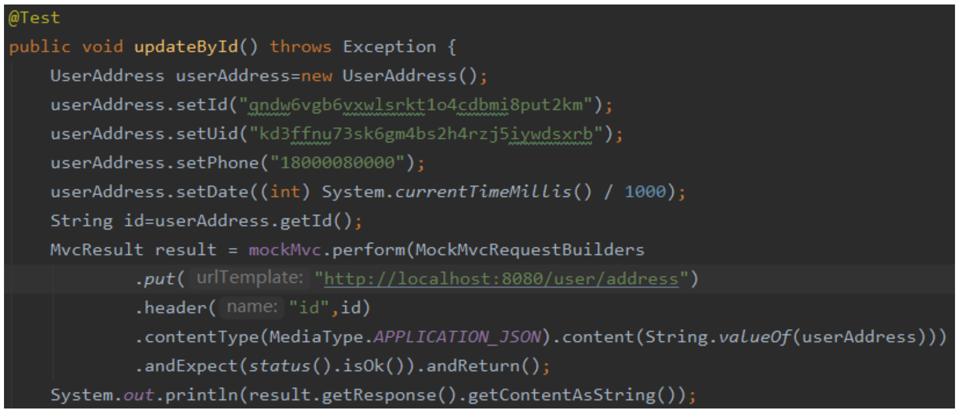
```
@Test
public void query() throws Exception {
    UserAddress userAddress=new UserAddress();
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    String user=userAddress.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .get( urlTemplate: "http://localhost:8080/user/address")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON)
            .content(String.valueOf(userAddress))).andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```



```
@Test
public void findById() throws Exception {
    UserAddress userAddress=new UserAddress();
    userAddress.setId("6kuweubglkhl9rewarridj2gyx8gqiwo");
    String id=userAddress.getId();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .get( urlTemplate: "http://localhost:8080/user/addressId")
            .header( name: "id",id)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userAddress.getId())))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```



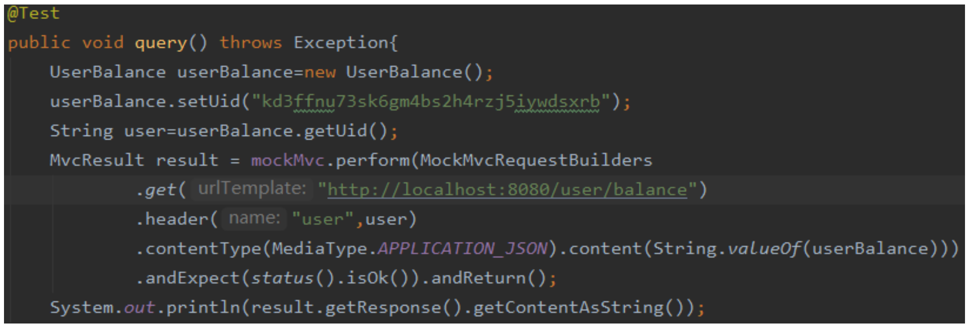
```
@Test
public void insert() throws Exception {
    UserAddress userAddress=new UserAddress();
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userAddress.setPhone("16000060000");
    userAddress.setName("Jiawei");
    userAddress.setCity("China");
    userAddress.setProvince("Guangdong");
    userAddress.setCity("Foshan");
    userAddress.setDist("Zengcheng");
    userAddress.setNote("No.1 Jianghe Road");
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    String user=userAddress.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/address")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userAddress)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```



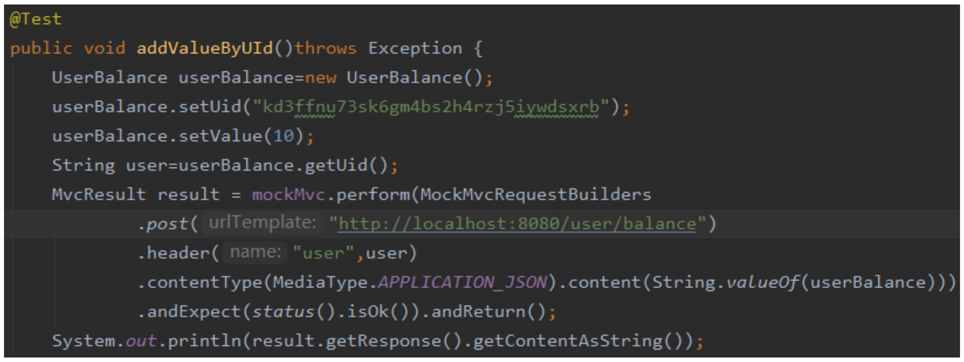
```
@Test
public void updateById() throws Exception {
    UserAddress userAddress=new UserAddress();
    userAddress.setId("gndw6vgb6vxwlsrkt1o4cdbmi8put2km");
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userAddress.setPhone("18000080000");
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    String id=userAddress.getId();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .put( urlTemplate: "http://localhost:8080/user/address")
            .header( name: "id",id)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userAddress)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

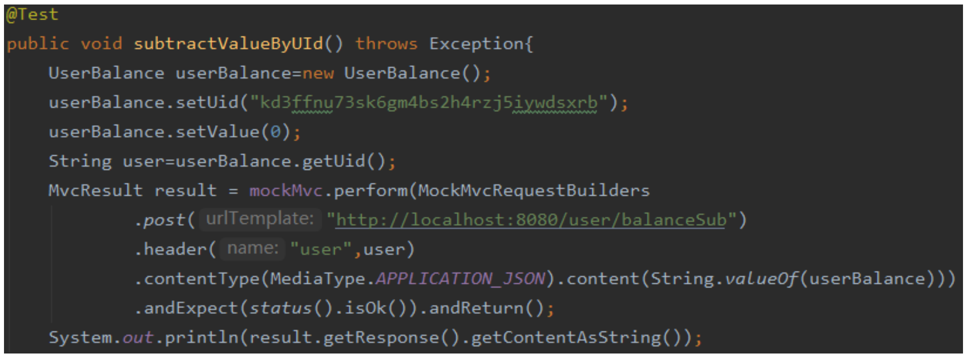
4. Test of UserBalanceController:

   Use mockMvc.perform() to execute the get/post/post request. Use MockMvcRequestBuilders.post("http://localhost:8080/user/balance") to construct these requests. .header() and .content() for passing parameters.



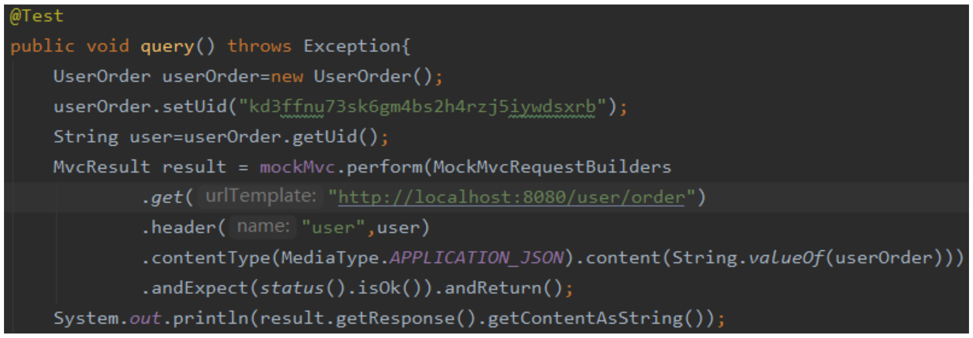
```
@Test
public void query() throws Exception{
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    String user=userBalance.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .get( urlTemplate: "http://localhost:8080/user/balance")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userBalance)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

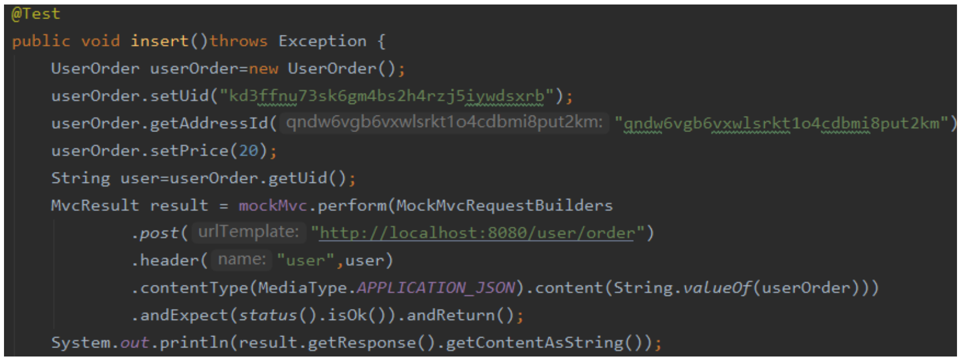

```java
@Test
public void addValueByUId()throws Exception {
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userBalance.setValue(10);
    String user=userBalance.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/balance")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userBalance)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```


```java
@Test
public void subtractValueByUId() throws Exception{
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userBalance.setValue(0);
    String user=userBalance.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/balanceSub")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userBalance)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

5. Test of UserOrderController:

   Use mockMvc.perform() to execute the get /post request. Use MockMvcRequestBuilders.post("http://localhost:8080/user/order") to construct these requests. .header() and .content() for passing parameters.

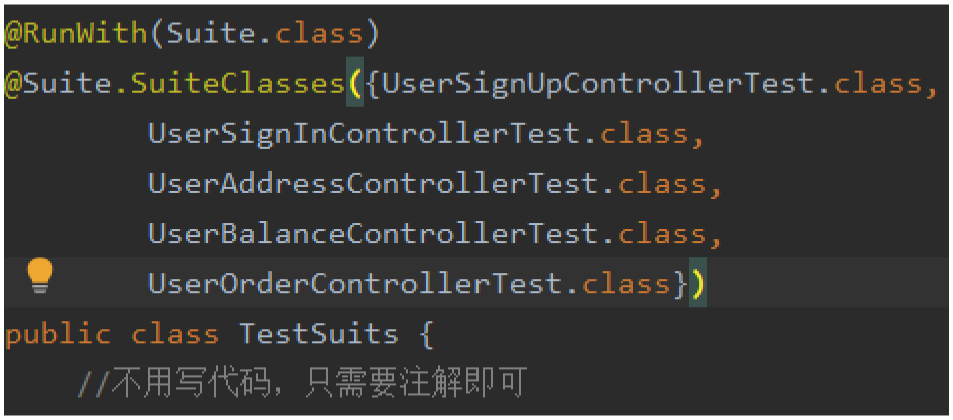

```java
@Test
public void query() throws Exception{
    UserOrder userOrder=new UserOrder();
    userOrder.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    String user=userOrder.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .get( urlTemplate: "http://localhost:8080/user/order")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userOrder)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```



```java
@Test
public void insert()throws Exception {
    UserOrder userOrder=new UserOrder();
    userOrder.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userOrder.getAddressId( qndw6vgb6vxwlsrkt1o4cdbmi8put2km: "qndw6vgb6vxwlsrkt1o4cdbmi8put2km")
    userOrder.setPrice(20);
    String user=userOrder.getUid();
    MvcResult result = mockMvc.perform(MockMvcRequestBuilders
            .post( urlTemplate: "http://localhost:8080/user/order")
            .header( name: "user",user)
            .contentType(MediaType.APPLICATION_JSON).content(String.valueOf(userOrder)))
            .andExpect(status().isOk()).andReturn();
    System.out.println(result.getResponse().getContentAsString());
```

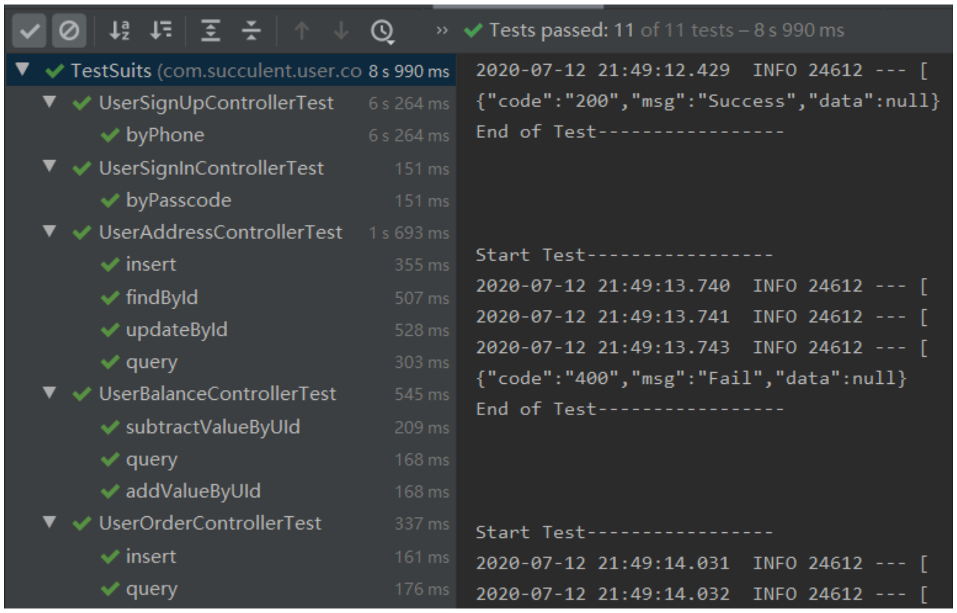
6. TestSuits:

   Use this class to run all test classes. Write the classes to be tested in .SuiteClasses().



```java
@RunWith(Suite.class)
@Suite.SuiteClasses({UserSignUpControllerTest.class,
        UserSignInControllerTest.class,
        UserAddressControllerTest.class,
        UserBalanceControllerTest.class,
        UserOrderControllerTest.class})
public class TestSuits {
    //不用写代码，只需要注解即可
```

7. Result of Tests:



```
TestSuits (com.succulent.user.co   8 s 990 ms
  UserSignUpControllerTest         6 s 264 ms
    byPhone                        6 s 264 ms
  UserSignInControllerTest             151 ms
    byPasscode                         151 ms
  UserAddressControllerTest        1 s 693 ms
    insert                             355 ms
    findById                           507 ms
    updateById                         528 ms
    query                              303 ms
  UserBalanceControllerTest            545 ms
    subtractValueByUId                 209 ms
    query                              168 ms
    addValueByUId                      168 ms
  UserOrderControllerTest              337 ms
    insert                             161 ms
    query                              176 ms
```

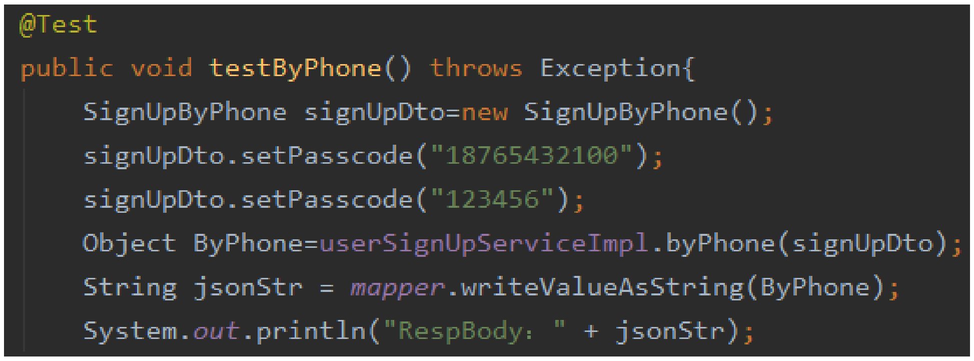
```
2020-07-12 21:49:12.429  INFO 24612 --- [
{"code":"200","msg":"Success","data":null}
End of Test----------------



Start Test----------------
2020-07-12 21:49:13.740  INFO 24612 --- [
2020-07-12 21:49:13.741  INFO 24612 --- [
2020-07-12 21:49:13.743  INFO 24612 --- [
{"code":"400","msg":"Fail","data":null}
End of Test----------------



Start Test----------------
2020-07-12 21:49:14.031  INFO 24612 --- [
2020-07-12 21:49:14.032  INFO 24612 --- [
```

7.2 Test of Service (Unit Test)

Find out the internal logic and function errors of the module.

1. Test of UserSignUpService:

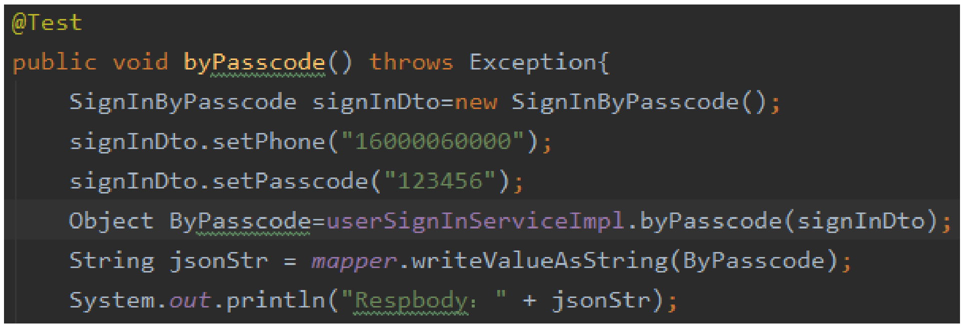
   Inject an instance of user account, call the corresponding interface, get and show the return result.



```
@Test
public void testByPhone() throws Exception{
    SignUpByPhone signUpDto=new SignUpByPhone();
    signUpDto.setPasscode("18765432100");
    signUpDto.setPasscode("123456");
    Object ByPhone=userSignUpServiceImpl.byPhone(signUpDto);
    String jsonStr = mapper.writeValueAsString(ByPhone);
    System.out.println("RespBody: " + jsonStr);
```

2. Test of UserSignInService:

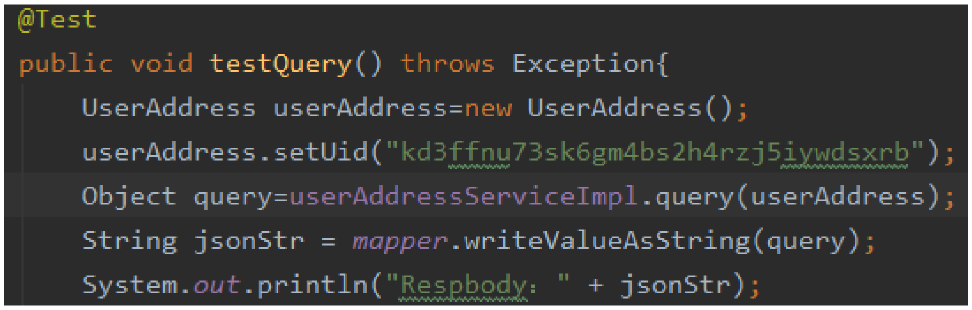
   Inject an instance of user log in account, call the corresponding interface, get and show the return result.



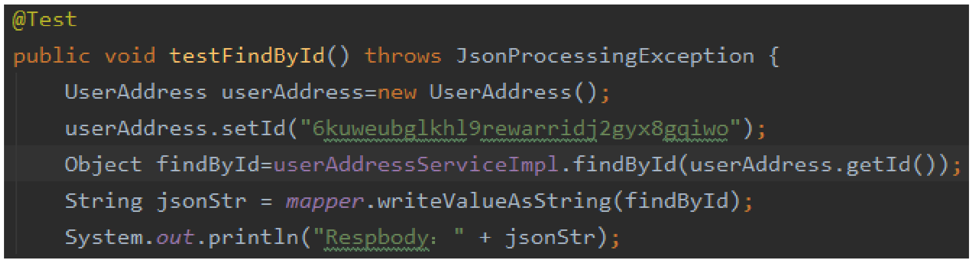
```java
@Test
public void byPasscode() throws Exception{
    SignInByPasscode signInDto=new SignInByPasscode();
    signInDto.setPhone("16000060000");
    signInDto.setPasscode("123456");
    Object ByPasscode=userSignInServiceImpl.byPasscode(signInDto);
    String jsonStr = mapper.writeValueAsString(ByPasscode);
    System.out.println("Respbody: " + jsonStr);
```

3. Test of UserAddressService:

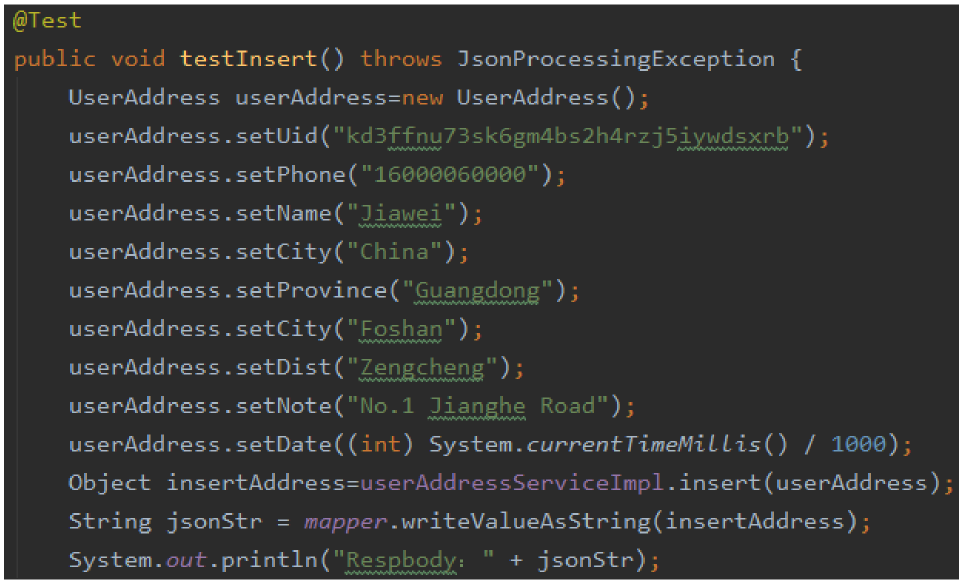
   Inject an instance of user address and update his address, call the corresponding interface, get and show the return result.

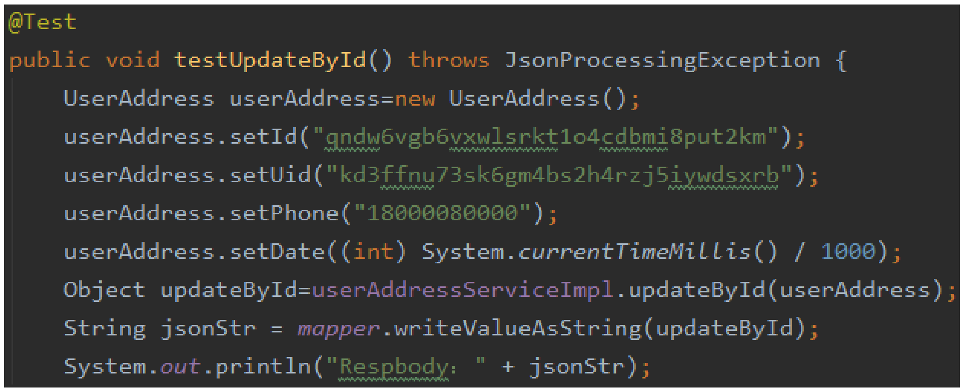


```java
@Test
public void testQuery() throws Exception{
    UserAddress userAddress=new UserAddress();
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    Object query=userAddressServiceImpl.query(userAddress);
    String jsonStr = mapper.writeValueAsString(query);
    System.out.println("Respbody: " + jsonStr);
```



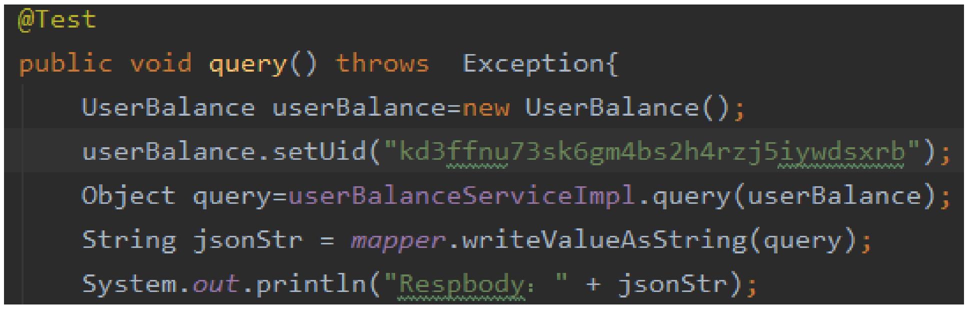
```java
@Test
public void testFindById() throws JsonProcessingException {
    UserAddress userAddress=new UserAddress();
    userAddress.setId("6kuweubglkhl9rewarridj2gyx8gqiwo");
    Object findById=userAddressServiceImpl.findById(userAddress.getId());
    String jsonStr = mapper.writeValueAsString(findById);
    System.out.println("Respbody: " + jsonStr);
```

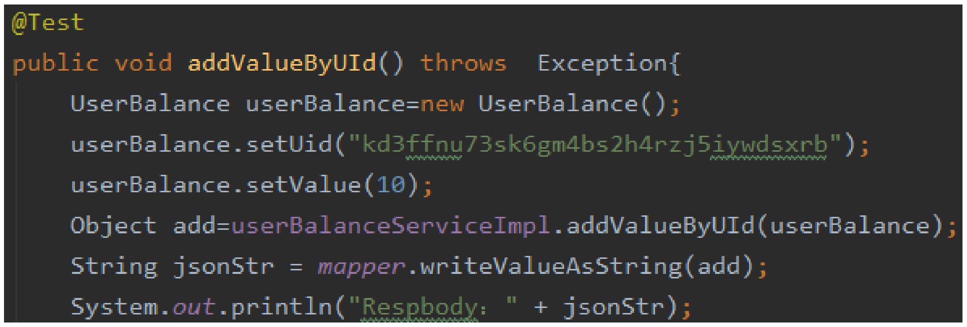

```
@Test
public void testInsert() throws JsonProcessingException {
    UserAddress userAddress=new UserAddress();
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userAddress.setPhone("16000060000");
    userAddress.setName("Jiawei");
    userAddress.setCity("China");
    userAddress.setProvince("Guangdong");
    userAddress.setCity("Foshan");
    userAddress.setDist("Zengcheng");
    userAddress.setNote("No.1 Jianghe Road");
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    Object insertAddress=userAddressServiceImpl.insert(userAddress);
    String jsonStr = mapper.writeValueAsString(insertAddress);
    System.out.println("Respbody: " + jsonStr);
```


```
@Test
public void testUpdateById() throws JsonProcessingException {
    UserAddress userAddress=new UserAddress();
    userAddress.setId("qndw6vgb6vxwlsrkt1o4cdbmi8put2km");
    userAddress.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userAddress.setPhone("18000080000");
    userAddress.setDate((int) System.currentTimeMillis() / 1000);
    Object updateById=userAddressServiceImpl.updateById(userAddress);
    String jsonStr = mapper.writeValueAsString(updateById);
    System.out.println("Respbody: " + jsonStr);
```

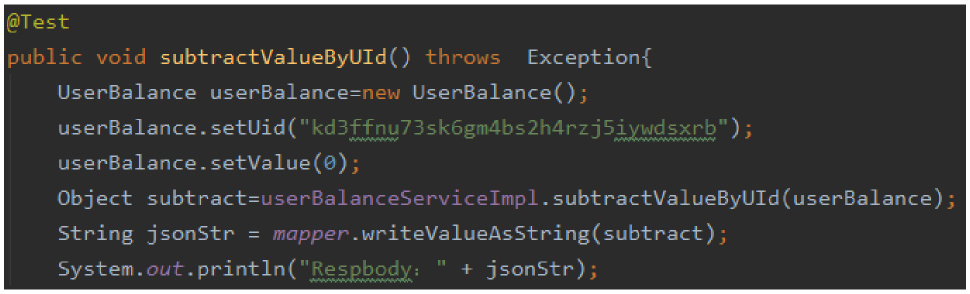
4. Test of UserBalanceService:

Inject an instance of user balance and update his balance, call the corresponding interface, get and show the return result.


```
@Test
public void query() throws  Exception{
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    Object query=userBalanceServiceImpl.query(userBalance);
    String jsonStr = mapper.writeValueAsString(query);
    System.out.println("Respbody: " + jsonStr);
```



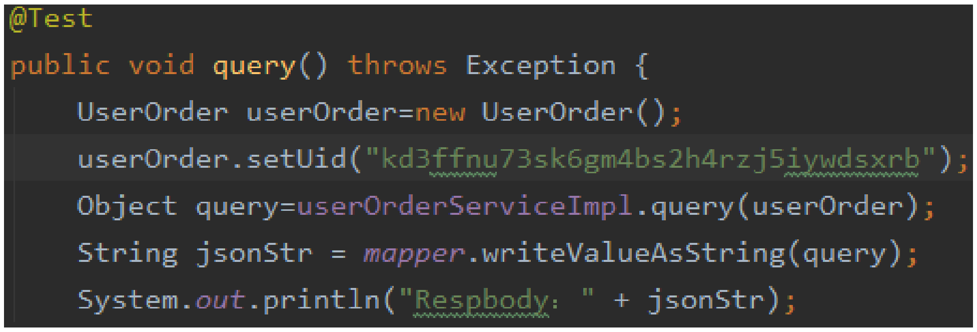
```java
@Test
public void addValueByUId() throws  Exception{
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userBalance.setValue(10);
    Object add=userBalanceServiceImpl.addValueByUId(userBalance);
    String jsonStr = mapper.writeValueAsString(add);
    System.out.println("Respbody: " + jsonStr);
```



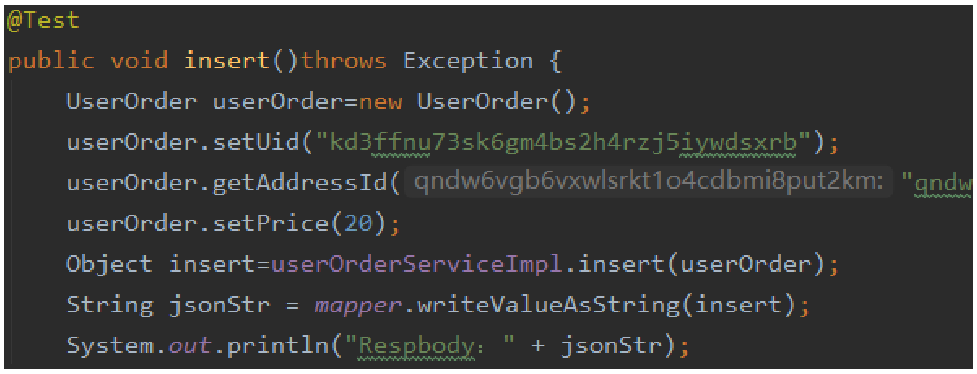
```java
@Test
public void subtractValueByUId() throws  Exception{
    UserBalance userBalance=new UserBalance();
    userBalance.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userBalance.setValue(0);
    Object subtract=userBalanceServiceImpl.subtractValueByUId(userBalance);
    String jsonStr = mapper.writeValueAsString(subtract);
    System.out.println("Respbody: " + jsonStr);
```

5. Test of UserOrderService:

   Inject an instance of user order, call the corresponding interface, get and show the return result.



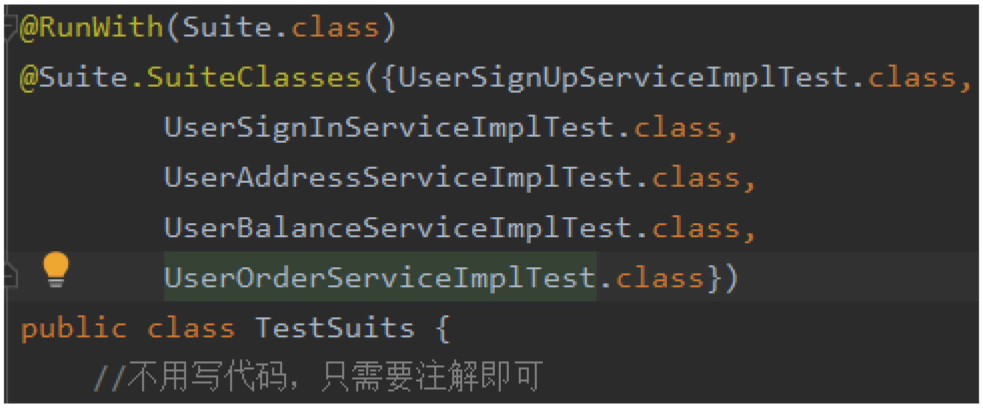
```java
@Test
public void query() throws Exception {
    UserOrder userOrder=new UserOrder();
    userOrder.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    Object query=userOrderServiceImpl.query(userOrder);
    String jsonStr = mapper.writeValueAsString(query);
    System.out.println("Respbody: " + jsonStr);
```



```java
@Test
public void insert()throws Exception {
    UserOrder userOrder=new UserOrder();
    userOrder.setUid("kd3ffnu73sk6gm4bs2h4rzj5iywdsxrb");
    userOrder.getAddressId( qndw6vgb6vxwlsrkt1o4cdbmi8put2km: "qndw
    userOrder.setPrice(20);
    Object insert=userOrderServiceImpl.insert(userOrder);
    String jsonStr = mapper.writeValueAsString(insert);
    System.out.println("Respbody: " + jsonStr);
```

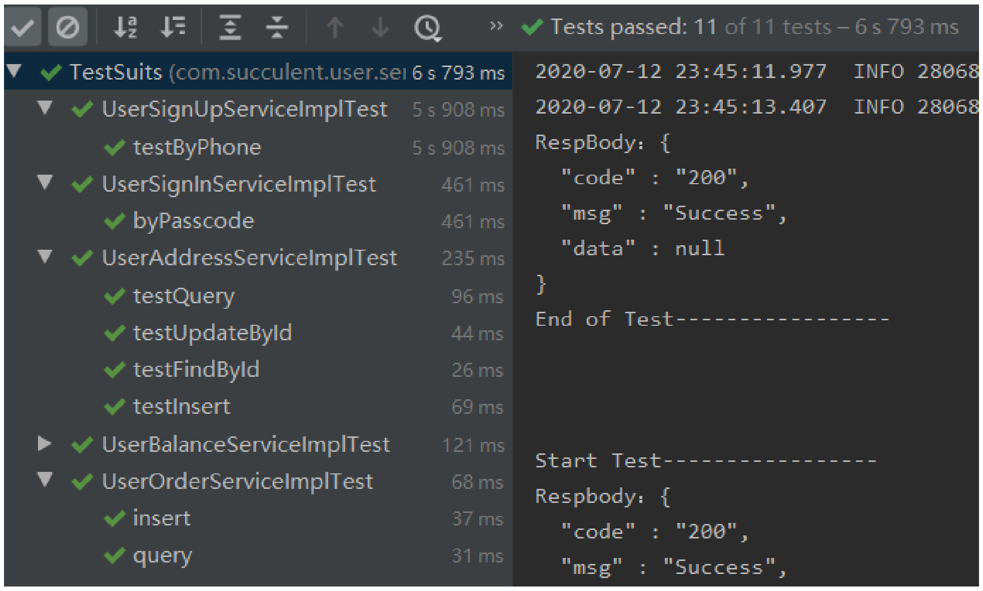
6. TestSuits:

Use this class to run all test classes. Write the classes to be tested in .SuiteClasses().



```
@RunWith(Suite.class)
@Suite.SuiteClasses({UserSignUpServiceImplTest.class,
        UserSignInServiceImplTest.class,
        UserAddressServiceImplTest.class,
        UserBalanceServiceImplTest.class,
        UserOrderServiceImplTest.class})
public class TestSuits {
    //不用写代码，只需要注解即可
```

7. Result of Tests:



```
Tests passed: 11 of 11 tests – 6 s 793 ms

TestSuits (com.succulent.user.sei       6 s 793 ms
  UserSignUpServiceImplTest             5 s 908 ms
    testByPhone                         5 s 908 ms
  UserSignInServiceImplTest               461 ms
    byPasscode                            461 ms
  UserAddressServiceImplTest              235 ms
    testQuery                              96 ms
    testUpdateById                         44 ms
    testFindById                           26 ms
    testInsert                             69 ms
  UserBalanceServiceImplTest              121 ms
  UserOrderServiceImplTest                 68 ms
    insert                                 37 ms
    query                                  31 ms
```

```
2020-07-12 23:45:11.977  INFO 28068
2020-07-12 23:45:13.407  INFO 28068
RespBody: {
  "code" : "200",
  "msg" : "Success",
  "data" : null
}
End of Test-----------------



Start Test-----------------
Respbody: {
  "code" : "200",
  "msg" : "Success",
```

8 Status of In-Use Implementation

When I left the company at the end of the internship period, the completion of the back-end development of the project is as follows:

No.	Description of objective	The extent to which this objective been achieved during the reporting period	The extent to which this objective been achieved during the project execution so far
1.	Project environment configuration and construction	Completed	Added some new contents: models and tables
2.	Database model and table building	Completed	Completed
3.	User registration and login function	Completed	Added some new functions: SMS verification code, etc.
4.	Functions corresponding to user's address, account balance and information	The main codes of these functions are completed	Completed

Table 1: Status of In-Use Implementation

9 Conclusion

Through participating in the design and back-end development of this online shopping mall system in this internship, I learned and mastered the popular web development technology in the industry. The following are the state of the art, the summary of this internship and the outlook of the future goal.

9.1 State of the Art

The traditional SSM and SSH frameworks require a lot of configuration work during system development. Especially in the early stage of software development, many XML configuration files need to be written to effectively integrate and correctly configure each part of the system. These seriously affect the efficiency of code development and raise the technical threshold.

The back-end of this online shopping mall is created using the system structure in this report which use Springboot instead of traditional tools. SpringBoot uses the concept of "contracting is better than configuration", so there is no need or little Spring configuration during the development process. All of it has effectively solved the problems of complicated configuration and low efficiency in the development process of the traditional SSM framework in the back-end development.

This report can help developers as a useful example when they are using Springboot in the back-end development of JavaWeb projects with SSM framework.

9.2 Summary of Internship:

In this internship, my main work in this project includes:

1. Participated in the analysis of the system functional demands of the online shopping mall project, detailed and confirmed some demands;
2. Participated in the construction of architecture and the overall framework of this online shopping mall project;
3. Developed the back end of the user’s module. There are user registration, user login, user order and user address modules;
4. Test the completed modules before leaving the company and inform the director of progress properly.

9.3 The Outlook of the Future Goal

In this internship, because of the impact of the coronavirus epidemic, the progress of the project is relatively slow. But, I still learned a lot of new knowledge and technology and deepened the understanding of web development.

For now, I am doing a job in a field related to IoT: Embedded development based on Hisilicon chip. And I am doing the application for Ph.D program in the field of IoT and will do some research in the future.

In a word, I quite enjoy this summer internship. It helps me to find out what's my favorite thing and inspired me a lot to work better in the future.

Here, I want to "Merci beaucoup" to my teachers: Pierre MARET, Olivier BOISSIER and Gauthier PICARD. Thank you for your careful guidance and patience, so that I can better optimize my report.

Thanks for your time.

Thank you.

References

[1]. PAN Shuang-shuang. "Innovative service platform of power business expansion for major clients based on the internet plus technology." *Power Demand Side Management* (2016).

[2]. Weng, Xingang , and L. Zhang . "Analysis of O2O Model's Development Problems and Trend." *Ibusiness* 7.1(2015):51-57.

[3]. Yu Yuanyuan [1]. "Design and Implementation of Online Mall Based on ASP.NET." Computer Products and Circulation 000.004(2018): P.162-162.

[4]. Xu Wen, and Gao Jianhua. "Research on Web Application Framework Based on Spring MVC and MyBatis." Microcomputer Application 07(2012):5-8+14.

[5]. Hai-Long, W. U. , and L. I. Guo-Ping . "Design of Housing Lease Management System Based on SpringBoot." Computer and Information Technology (2019).

[6]. Liang Dan. Application of UML in object-oriented relational database[J]. Shanxi Science and Technology, 2009(3): 34-35.

[7]. Yang Kaizhen, Zhou Jiwen, Liang Huahui, Tan Maohua. Java EE Internet lightweight framework integrated development of SSM framework (Spring MVC + Spring + MyBatis) and Redis implementation [M]. Beijing: Electronic Industry Press, 2017.

[8]. Zhang Liangfeng, Lin Yimin. Design and Implementation of Haitian Online Mall Based on Integration of Three SSM Frameworks[J]. Software Engineering and Applications, 2017, 6(6): 240-255.