SUSTech Southern University of Science and Technology

CS307 Principles of Database Systems

Project 2 Report

Database of SUSTC

11911612 Haoyu Wang
12112011 YiYu Liu

class 1 group 1
December 23, 2022

Department of Computer Science and Engineering
南方科技大学计算机科学与工程系

SUSTech Southern University of Science and Technology

Contents

1. Contribution
2. Task 1: Database Design
3. Task 2: Basic APIs implementation And Analyses
4. Task 3: Advanced APIs And Frontend Design
5. Summary

1. Contributions

Haoyu Wang:

1. Database design
2. Design and draw ER diagram
3. The implementation of interfaces
4. Design and implement front end and servlet. (using Tomcat)
5. Write the report

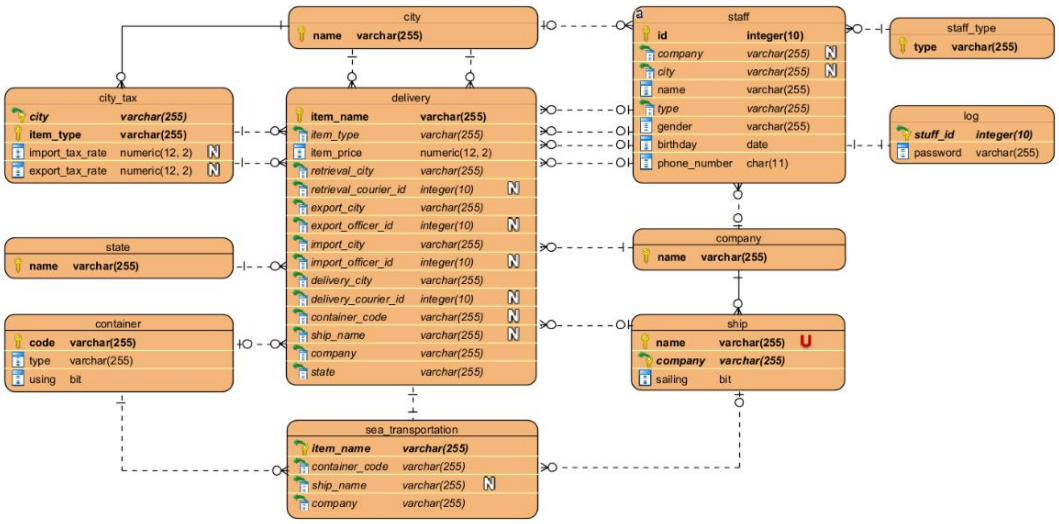
Yiyu Liu:

1. Database design
2. Import data into database
3. Modify DAO components and implement the JAVA project architecture
4. API tests and optimization
5. Write the report

2. Task 1: Database Design

2.1 ER Diagram and Database Design of project 2:

using Visual Paradigm



(for each attribute, sign N means it can be null, sign U means it is unique)

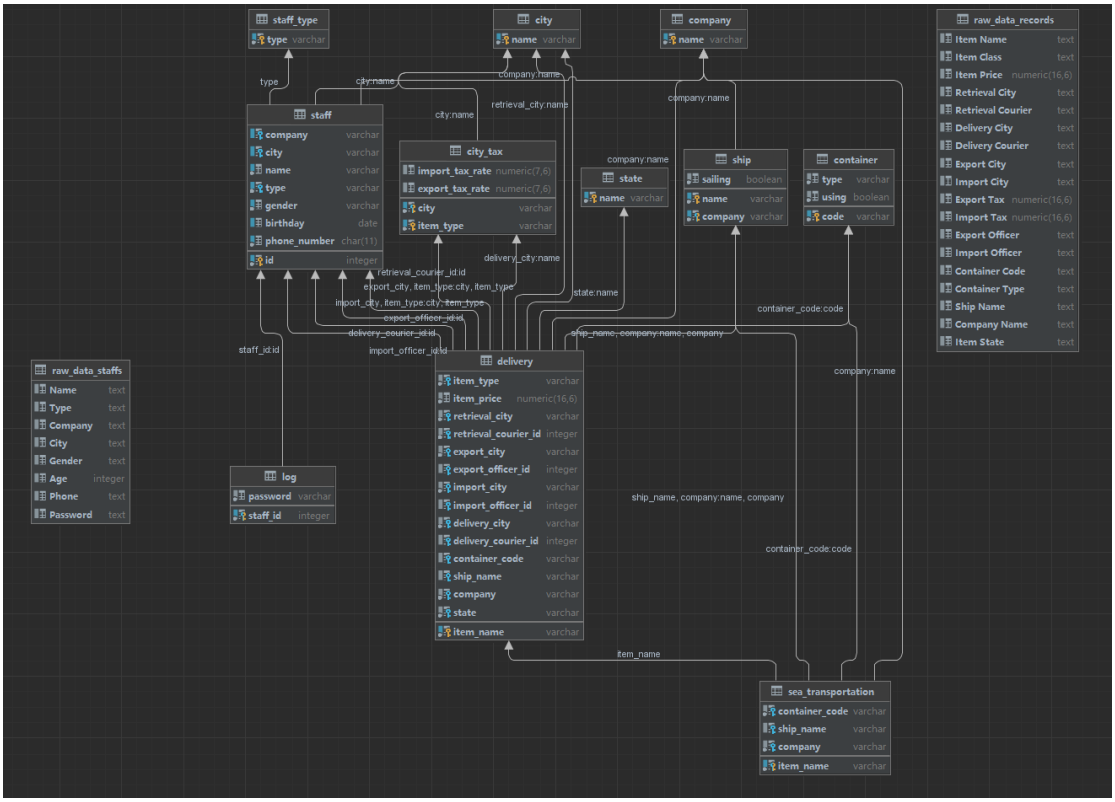
The Design of Database:

1. Two Primary Key in table “city tax” and “ship”: confirm the consistency of database
2. The field “using” of table “container” and the field “sailing” of table ship: they are extra fields from project 1
3. Table “sea transportation”: This is a dynamic table, which records item and its current container and ship. When an item is packed into a container, one row will be inserted into table. When an item (or container) is loaded onto a ship, one row will be updated. And when an item is unpacked, one row will be deleted from table.
表 “sea transportation” 是一个动态调整的表，它会随着物品的放入集装箱、上船、卸载而不断变化。当物品进箱时，会做 insert 操作，物品和它的集装箱上船的时候，会做 update 操作，物品被卸货的时候，还会做 delete 操作。在从装箱到船运到卸货的整个过程，我们主要都在对该表进行操作，并通过触发器更新其他相关的属性。
4. Github 上给出了该查询 container 是否被使用以及 ship 是否在航行的方式，都需要在 delivery 中使用复杂查询。而我们的设计中通过维护 container 表中的 using 字段和 ship 表中的 sailing 字段，极大简化了这两者的查询。

7. How can one obtain a shipping or occupying status of a ship or a container?

- Via SQL statements that look up for item(s) of `Shipping` state with given ship or items of `Packing to Container` -- `Unpacking from Container` states with given container.

2.2 Visualization of DataGrip



2.3 Descriptions on Tables

1. **Basic information of tables**
   1) “raw_data_records”: original data from records.csv.
   2) “raw_data_staffs”: original data from staffs.csv.
   3) “city”: city names.
   4) “city_tax”: for port city, it records export and import tax rates corresponds to item class
   5) “state”: state of each record.
   6) “container”: code, type, whether it is using.
   7) “ship”: name, company, whether it is sailing.
   8) “company”: company names.
   9) “staff”: id, company, city, name, staff types, gender, birthday, phone number.
   10) “staff_type”: all staff types in “SUSTC Department Manager”, “Courier”, “Company Manager”, “Seaport Officer”.
   11) “log”: staffs with their passwords.
   12) “delivery”: information of each record.
   13) “sea_transportation”: information of records that states are during “Packing to Container” and “Unpacking from Container”.

2. **Specifications on tables**

1) Attributes “passwords” are excluded in the table “staff”. Rather than recording in “staff”, we use another table “log” to record passwords due to authorization management. The table “log” can only be accessed by user “LogChecker” and “SUSTC Department Manager”.
表“staff”不包含字段“password”。由于权限管理需要，相比于将密码放入表“staff”内，我们使用另一张表“log”去记录密码。该表仅能被用户“LogChecker”和“SUSTC Department Manager”访问。

2) Table “sea transportation” is a dynamic table. We design it in order to manage state transferring of delivery records, which has been introduced in The Design of Database.

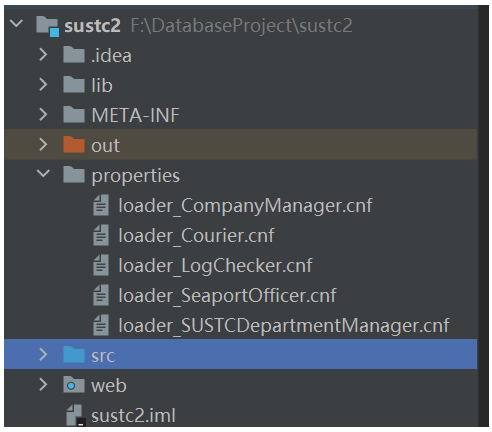
3) We use id to specify different staffs. However, in this project, each staff has a unique name. Hence in sql queries we may only use the condition of identical names to confirm sameness.

2.4 User Authorization Management

We have five users in total:

```
-- role
create user LogChecker with password '111111';
create user Courier with password '111111';
create user CompanyManager with password '111111';
create user SeaportOfficer with password '111111';
create user SUSTCDepartmentManager with password '111111';
```

When the constructor of class “DBManipulation” is executed, these five users will be created. And their login information, including host, database user name and password, will be stored in the cnf files, which are in an auto-created folder named properties.



There are user names and corresponding authorization:

1. **Log Checker：**

Log Checker is used to check whether the Log Information is correct or not.

```
-- LogChecker
GRANT SELECT, INSERT, UPDATE ON TABLE staff,log TO LogChecker;
```

2. **Courier**

Courier can insert new item into table delivery and update the item state in table delivery. He also can select some necessary information.

```
-- Courier:
GRANT SELECT, INSERT, UPDATE ON TABLE delivery TO Courier;
GRANT SELECT ON TABLE city,city_tax,staff,state TO Courier;
```

3. **Company Manager**

Company Manager can operate containers and ships and set item state in table delivery. He also has all privilege on table sea transportation, which has been introduced below.

```
-- CompanyManager:
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE sea_transportation TO CompanyManager;
GRANT SELECT, UPDATE ON TABLE delivery,container,ship,staff TO CompanyManager;
GRANT SELECT ON TABLE city,city_tax,company,staff_type,state TO CompanyManager;
```

4. **Seaport Officer**

Seaport Officer can only determine whether the export or import checking of delivery is successful or not.

```
-- SeaportOfficer:
GRANT SELECT, UPDATE ON TABLE delivery TO SeaportOfficer;
GRANT SELECT ON TABLE staff TO SeaportOfficer;
```

5. **SUSTC Department Manager**

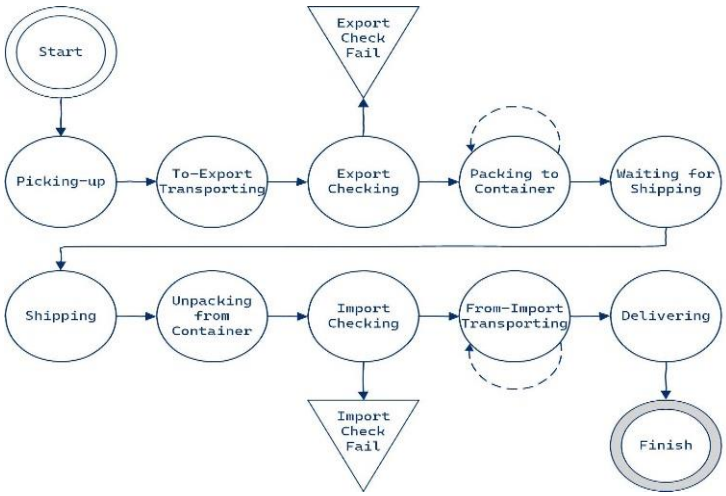
SUSTC department manager can select all information including staff password but do not have “insert”, “update” or “delete” authority. We think it is reasonable.

```
-- SUSTCDepartmentManager
GRANT SELECT ON TABLE city,city_tax,company,container,delivery,log,sea_transportation,ship,staff,staff_type,state
    TO SUSTCDepartmentManager;
```

2.5 Trigger

We have four triggers in total. For convenience, we name each trigger by its function. All of these triggers have been set as “before operation” and “for each row”. Detail:

1. **update delivery:**
   a) guarantee the state changed as below.




```
case NEW.state
    when 'Picking-up'
        then RAISE EXCEPTION 'failed';
    when 'To-Export Transporting'
        then if OLD.state ...;
    when 'Export Checking'
        then if OLD.state ...;
    when 'Export Check Fail'
        then if OLD.state ...;
    when 'Packing to Container'
        then if OLD.state ...;
             raise exception 'failed';
    when 'Waiting for Shipping'
        then if OLD.state ...;
    when 'Shipping'
        then if OLD.state ...;
    when 'Unpacking from Container'
        then if OLD.state ...;
    when 'Import Checking'
        then if OLD.state ...;
    when 'Import Check Fail'
        then if OLD.state ...;
    when 'From-Import Transporting'
        then if OLD.state ...;
    when 'Delivering'
        then if OLD.state ...;
    when 'Finish'
        then if OLD.state ...;
end case;
```

b) When item state is “Packing to Container”:

```
when 'Packing to Container'
    then if OLD.state like 'Export Checking' then
        return new;
        elseif OLD.state like 'Packing to Container' then
            select "using"
            from container
            where code like NEW.container_code
            into container_using;
            if container_using is false then
                delete
                from sea_transportation
                where item_name like old.item_name;
                update container
                set "using" = true
                where code like NEW.container_code;
                insert into sea_transportation
                values (new.item_name...[3 more]);
                return new;
            end if;
        end if;
        raise exception 'failed';
```

i. Check if the container using field is false;

```
select "using"
from container
where code like NEW.container_code
into container_using;
if container_using is false then
```

ii. Then we need to do 3 operations, delete, update, insert:

```
if container_using is false then
    delete
    from sea_transportation
    where item_name like old.item_name;
    update container
    set "using" = true
    where code like NEW.container_code;
    insert into sea_transportation
    values (new.item_name, new.container_code, null, new.company);
    return new;
end if;
```

Firstly, delete from table “sea transportation”. It is used to deal with this item have been packed into another container before. Secondly, set container field using is true. Thirdly, insert a new record into table “sea transportation”

当物品装载进一个集装箱时，我们会在表“sea transportation”中新建一个记录同时更新集装箱状态。但是这个物品有可能发生从集装箱 A 换到集装箱 B 的情况。这个时候，我们就需要先删除之前在表“sea transportation”中的记录，再重新插进去一个新的记录，而集装箱 A 的状态将通过触发器“delete trigger”去更新。

c) When new item state is “From-Import Transporting:

There are two cases. The seaport officer can change state from “Import Checking” into

"From-Import Transporting". And delivery courier can just update his id into the record in table "delivery" and keep the state the same as "From-Import Transporting".

```
when 'From-Import Transporting'
    then if OLD.state like ('Import Checking' or 'From-Import Transporting') and
            OLD.delivery_courier_id is null then
        return new;
    else
        raise exception 'failed';
    end if;
```

2. update ship

```
BEGIN
    if old.sailing is false and new.sailing is true
    then
        update delivery
        set state = 'Shipping'
        where ship_name like old.name
          and state like 'Waiting for Shipping';
        return new;
    end if;
    if old.sailing is true and new.sailing is false
    then
        return new;
    end if;
    RAISE EXCEPTION 'something wrong';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION '(%)', SQLERRM;
END;
```

a) The important point of field sailing in table "ship" is that when a company manager ask a ship to sail, all of the items in this ship should change their state from "Waiting for Shipping" into "Shipping".
开船的时候，通过该触发器更新此时船上所有货物的状态为"Shipping"

b) A ship can only update sailing from false into true or from true into false. Any other update is illegal.

3. update sea transportation

```
BEGIN
    if (old.ship_name is null and new.ship_name is not null) then
        update delivery
        set (state, ship_name) = ('Waiting for Shipping', new.ship_name)
        where item_name like new.item_name;
        return new;
    end if;
    return null;
END;
```

It only used to update the ship name when a container is loaded onto a ship.
该触发器在集装箱上船的时候被触发。这一步骤体现了在数据库设计中使用表"sea transportation"的优势。如果不考虑表 delivery，由于集装箱根本没有记录装载了哪个物品，也不知道自己会被装到哪个船只上；而船也并不知道自己装载，我们去维护整个流程非

常麻烦，需要反复去五十万行的 delivery 中查找。而表 sea transportation 完美解决了这个问题，它给出了物品，集装箱，船只的实时对应关系，并且通过完备的触发器保证了数据库中各表数据的一致性，在结构上非常简洁清晰。

4. delete sea transportation

```
if old.ship_name is not null then
    update delivery
    set state = 'Unpacking from Container'
    where item_name like old.item_name;
    select count(*) from sea_transportation where ship_name like old.ship_name into count;
    if (count = 1) then
        update ship set sailing = false where name like old.ship_name;
    end if;
end if;
update container
set "using" = false
where code like old.container_code;
return old;
```

When sea transportation needs to be updated, there are two cases:

a) an item is moved from container A into container B, whose state must be “Packing to Container”

b) an item is unpacked from the ship and container

For case a, we only need to change the container using from true into false;

For case b, besides that operation, we also need to set the item state into “Unpacking from Container”. What’s more, if this ship has no item on it, the sailing filed will be changed into false.

需要从表“sea transportation”删除记录有两种情况。一种情况是，物品从集装箱 A 改到集装箱 B，之前已经介绍过了，这时只需要更新集装箱 A 的 using 为 false。另一种情况是，一个物品要被卸载。我们再接口中去实现 unloadItem 方法时，仅仅会从执行从 sea transportation 中删除记录这一个操作。因此 item 在表“delivery”中状态的改变，也要由这个触发器去执行。

3. Task 2: Basic APIs implementation And Analyses

3.1 Basic APIs implementation structures

In our project, all basic APIs are implemented at bottom through DAO(Database Access Objects). When a query is put forward (specifically, an interface method of API is called), after checking user's authorization, our system calls the service class needed, which then calls its subordinate DAO class to access the table needed in the database, thus to achieve corresponding instructions.
All of these interface methods are placed in class DBManipulation.
在我们的项目中，所有的基础 API 在底层都是通过 DAO（数据库访问对象）实现的。当提出指令时（即调用 API 的接口方法），我们的系统在核查用户权限后，将调用所需的服务器类，服务类再调用其下属的 DAO 类来访问数据库中所需的表，以实现相应的指令。
所有这些接口方法都在类 DBManipulation 中。

DBManipulation.java

```
public StaffInfo getStaffInfo(LogInfo logInfo, String staff_name) {
    if (logInfo.type() == LogInfo.StaffType.SustcManager && logService.check_logInfo(logInfo)) {
        CON = getConnection( load_cnf: "loader_SUSTCDepartmentManager.cnf");
        Staff staff = staffService.getStaff( loader_cnf: "loader_SUSTCDepartmentManager.cnf", staff_name);
        Calendar ca = Calendar.getInstance();
        ca.setTime(staff.getBirth_year());
        int year = ca.get(Calendar.YEAR);
        String password = logService.getPassword( loader_cnf: "loader_SUSTCDepartmentManager.cnf", staff.getId());
        LogInfo l = new LogInfo(staff.getName(), LogInfo.StaffType.valueOf(staff.getType().replaceAll( regex: "",
        closeResource(CON);
        return new StaffInfo(l, staff.getCompany(), staff.getCity(), staff.getGender().equals("female"),  age: 202
    }
    return null;
}
```

StaffServiceImpl.java

```
public Staff getStaff(String loader_cnf, String staff_name) {
    Staff[] staffs = staffDao.getStaff(loader_cnf, staff_name);
    if (staffs.length == 1)
        return staffs[0];
    return null;
}
```

StaffDaoImpl.java

```
public Staff[] getStaff(String loader_cnf, String staff_name) {
    String sql;
    ArrayList<Staff> staffs = new ArrayList<>();
    try {...} catch (Exception e) {...} finally {...}
    return staffs.toArray(new Staff[0]);
}
```

3.2 Implementation of DAOs

Queries are achieved in DAOs mainly by executing sql statements.

For method "$import()", we simply get connection of root user, then execute sql statements to import data to the target table accessed by this DAO object.

```java
@Override
public void importData(boolean verbose) {
    Statement stmt = null;
    try {
        // connection
        connection = PostgresqlUtil.getConnectionRoot();
        System.out.println("CityDaoImpl importData() start");

        // Drop table
        stmt = connection.createStatement();
        stmt.execute( sql: "truncate table city cascade;");

        // import from raw_data
        String sql = """
                insert into city(name)
                select * from (
                    select "Retrieval City" from raw_data_records UNION
                    select "Delivery City" from raw_data_records UNION
                    select "Export City" from raw_data_records UNION
                    select "Import City" from raw_data_records
                    ) as t;
                """;
        stmt.execute(sql);
        connection.commit();

        System.out.println("Load Data into table city SUCCESSFULLY.");
    } catch (Exception e) {...} finally {...}
}
```

For methods of different staffs, the specific terms needed to fill in the statements are passed in as parameters. The user info is also among that.
For update and delete operations, it simply executes sql statements, and for insert operations, we use a result set to store selection result, and then, after processing, return it as a piece of info. Here are three examples:

CityTaxDaoImpl.java

```java
public CityTax[] getTaxRate(String loader_cnf, String city, String type) {
    ArrayList<CityTax> cityTaxes = new ArrayList<>();
    String sql;
    try {
        //connection = PostgresqlUtil.getConnection(loader_cnf);
        sql = "select * from city_tax where city like ? and item_type like ?";
        ResultSet resultSet = execute_query(CON, sql, new String[]{city, type});
        if (verbose)
            System.out.println("get data from table city_tax successfully");
        while (resultSet.next()) {
            cityTaxes.add(new CityTax(resultSet.getString( columnIndex: 1),
                    resultSet.getString( columnIndex: 2),
                    resultSet.getDouble( columnIndex: 3),
                    resultSet.getDouble( columnIndex: 4)));
        }
    } catch (Exception e) {...} finally {...}
    return cityTaxes.toArray(new CityTax[0]);
}
```

ShipDaoImpl.java

```java
@Override
public int getCount(String loader_cnf) {
    String sql;
    int count = -1;
    try {
        connection = PostgresqlUtil.getConnection(loader_cnf);
        sql = "select count(name) from ship";
        ResultSet resultSet = execute_query(connection, sql, params: null);
        while (resultSet.next())
            count = resultSet.getInt( columnIndex: 1);
        if (verbose)
            System.out.println("get ship count from table ship successfully");
    } catch (Exception e) {...} finally {...}
    return count;
}
```

SeaTransportationDaoImpl.java

```java
public boolean updateShip_by_containerCode(String loader_cnf, String container_code, String ship_name) {
    String sql;
    boolean flag = true;
    try {
        connection = PostgresqlUtil.getConnection(loader_cnf);
        sql = "update sea_transportation set ship_name = ? where container_code like ?";
        execute_update(connection, sql, new String[]{ship_name, container_code});
        if (verbose)
            System.out.println("update ship_name from table sea_transportation successfully");
    } catch (Exception e) {...} finally {...}
    return flag;
}
```

3.3 Optimization of APIs on running time

During out test, we found that the mainly time cost is the connection step. Each time of connection will cost about 40 ms.

For example, the method getItemInfo() in the class DBManipulation needs to call totally 10 getter methods of its service class with user info as parameters (8 getters in itself, 2 in the submethod check_logInfo()), which means if we do connections in getter methods, it will take more than 400 ms on getting connection, easily ending up in time limit exceeded. In fact, the test result of executing time of it is about 800-1200 ms.

Using a member of connection in the class DBManipulation or using a global connection can solve this problem (variable CON). Because each interface method in DBManipulation depends on a single specific user, we only need to get connection once. This ends up in about 200-300 ms.

Although, we generally expect to let service classes to manage connections to the database, here we quit this principal for efficiency.

在测试中我们发现，主要的时间成本是连接数据库，每次连接将花费约 40 ms。在原始设计中，为保证每次对数据库操作的独立性，我们在一个 DBManipulation 内可能进行多次数据库连接。下面以 getItemInfo()方法进行举例。

getItemInfo()方法需要调用 10 个以用户信息为参数且需要连接数据库的 getter 方法去进行数据查询，这意味着如果我们在 getter 方法中进行连接，将花费 400 多毫秒来获得连接，很容易超时。其实际执行时间的测试结果为 800-1200 ms。

使用 DBManipulation 类中的成员连接或使用全局连接可以解决这个问题。因为 DBManipulation 中的每个接口方法都依赖于单个特定用户，所以我们只需要获得一次连接。这样花费的总时间为 200-300 ms。

节约了约 75%的时间消耗。

具体**数据**如图所示：

getItemInfo() 中仅连接 Connection 每次所花时间(ms)		
调用方法	连接次数	执行时间(ms)
check_logInfo()	2	96
getConnection()	1	40
getDelivery()	1	48
RetrievalDeliveryInfo()	1	52
RetrievalDeliveryInfo()	1	48
ImportExportInfo()	2	86
ImportExportInfo()	2	90
		460

如图：在该方法开始部分使用全局变量 CON 连接数据库，下面各方法在具体的实现过程中，均直接使用了该全局变量：

```
@Override
public ItemInfo getItemInfo(LogInfo logInfo, String item_name) {
    if (logInfo.type() == LogInfo.StaffType.SustcManager && logService.check_logInfo(logInfo)) {

        CON = getConnection( load_cnf: "loader_SUSTCDepartmentManager.cnf");

        Delivery deliveryRecord = deliveryService.getDelivery( loader_cnf: "loader_SUSTCDepartmentManager.cnf", item_name);
        if (deliveryRecord != null) {

            ItemInfo.RetrievalDeliveryInfo retrieval = new ItemInfo.RetrievalDeliveryInfo(deliveryRecord.getRetrieval_city(),
                    staffService.getStaffName( loader_cnf: "loader_SUSTCDepartmentManager.cnf", deliveryRecord.getRetrieval_courier_id()));

            ItemInfo.RetrievalDeliveryInfo delivery = new ItemInfo.RetrievalDeliveryInfo(deliveryRecord.getDelivery_city(),
                    staffService.getStaffName( loader_cnf: "loader_SUSTCDepartmentManager.cnf", deliveryRecord.getDelivery_courier_id()));

            ItemInfo.ImportExportInfo export = new ItemInfo.ImportExportInfo(deliveryRecord.getExport_city(),
                    staffService.getStaffName( loader_cnf: "loader_SUSTCDepartmentManager.cnf", deliveryRecord.getExport_officer_id()),
                     tax: cityTaxService.getExportTaxRate( load_cnf: "loader_SUSTCDepartmentManager.cnf",
                            deliveryRecord.getExport_city(), deliveryRecord.getItem_type()) * deliveryRecord.getItem_price());

            ItemInfo.ImportExportInfo $import = new ItemInfo.ImportExportInfo(deliveryRecord.getImport_city(),
                    staffService.getStaffName( loader_cnf: "loader_SUSTCDepartmentManager.cnf", deliveryRecord.getImport_officer_id()),
                     tax: cityTaxService.getImportTaxRate( load_cnf: "loader_SUSTCDepartmentManager.cnf",
                            deliveryRecord.getImport_city(), deliveryRecord.getItem_type()) * deliveryRecord.getItem_price());

            return new ItemInfo(
                    deliveryRecord.getItem_name(),
                    deliveryRecord.getItem_type(),
                    deliveryRecord.getItem_price(),
                    getItemState_from_String(deliveryRecord.getState()),
                    retrieval,
                    delivery,
                    $import,
                    export
            );
        }

        closeResource(CON);
    }
    return null;
}
```

附：测试环境

系统环境：Windows

其他配置如下图所示：

设备名称	LAPTOP-9LCQ1TF1
处理器	AMD Ryzen 7 5800U with Radeon Graphics 1.90 GHz
机带 RAM	16.0 GB (15.4 GB 可用)
设备 ID	DE03266E-C972-4443-A47E-3716FAF0BA2C
产品 ID	00342-36298-08907-AAOEM
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	为 20 触摸点提供笔和触控支持

4. Task 3: Advanced APIs And Frontend Design

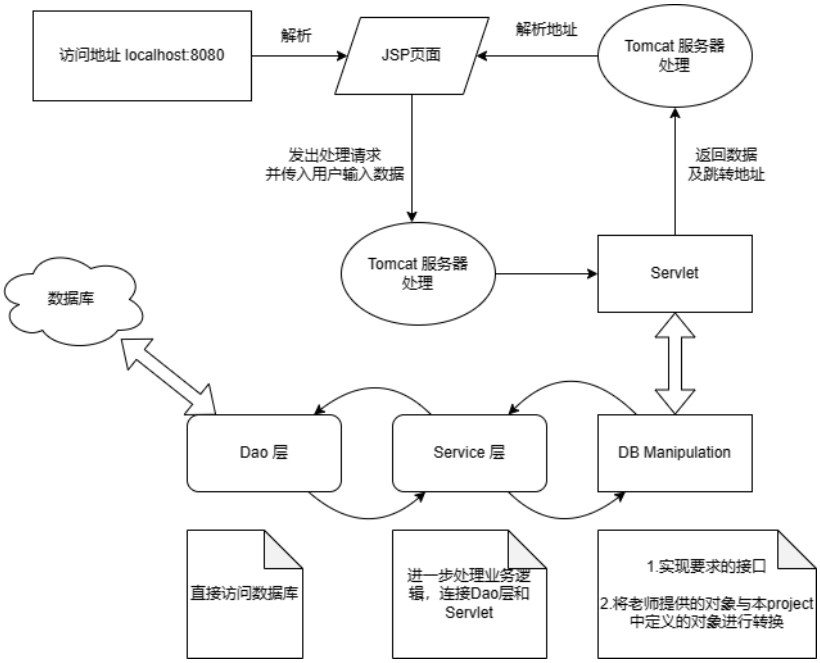
注：为便于表述及阅读，本部分完全使用中文书写

在本次 project 中，我们使用 Tomcat 实现了 java web 应用服务器。我们惊叹于 jsp 能够内嵌 java 代码的强大功能和 Tomcat 提供的完整框架结构。学习了在 jsp 中使用 form 表单提交参数传递给后端的功能，学会了利用函数进行复杂操作，了解了 jsp 或者说 html 中各个控件的功能和实现。下面将给出本次 project 前后端交互的实现细节：

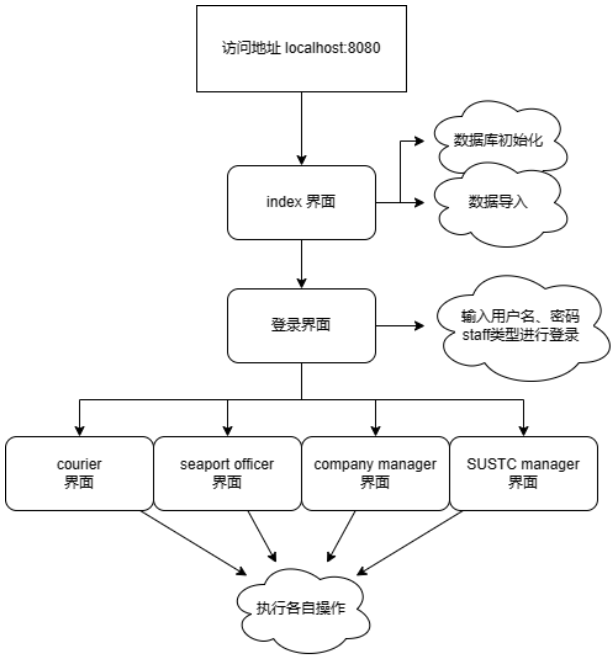
4.1 概述

使用 Tomcat 实现 java web 应用服务器。将 JSP 页面的请求通过服务器发给 Servlet 层，对请求进行响应并调取 DBManipulation 中的相应方法，最终将结果返回到 JSP 页面中展示出来。具体流程如下图所示：

（图片是根据我学了三天的 Tomcat 所理解的知识画出的，可能有部分不够精确的地方）



4.2 操作的逻辑流程如下图所示：

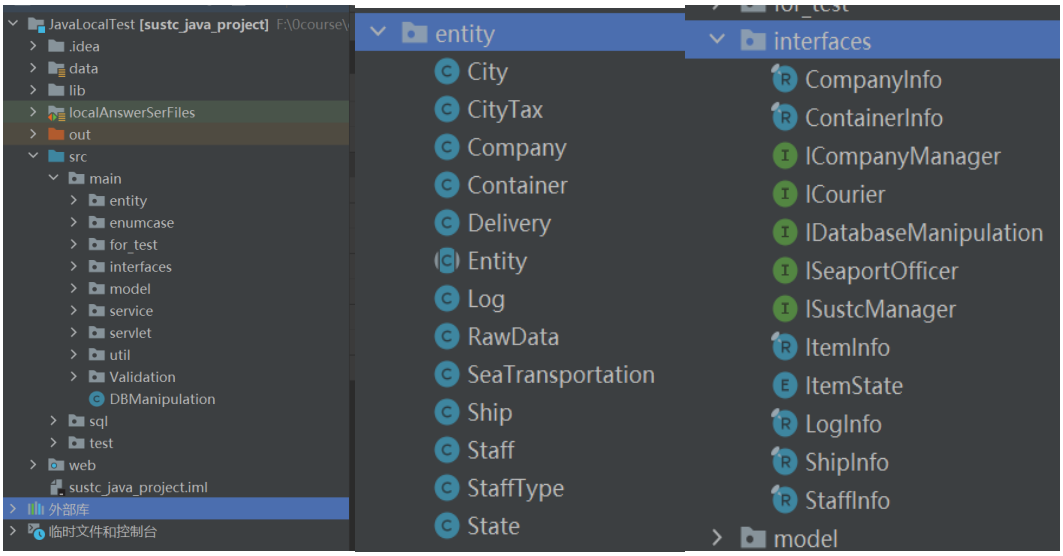


4.3 Java 项目结构：

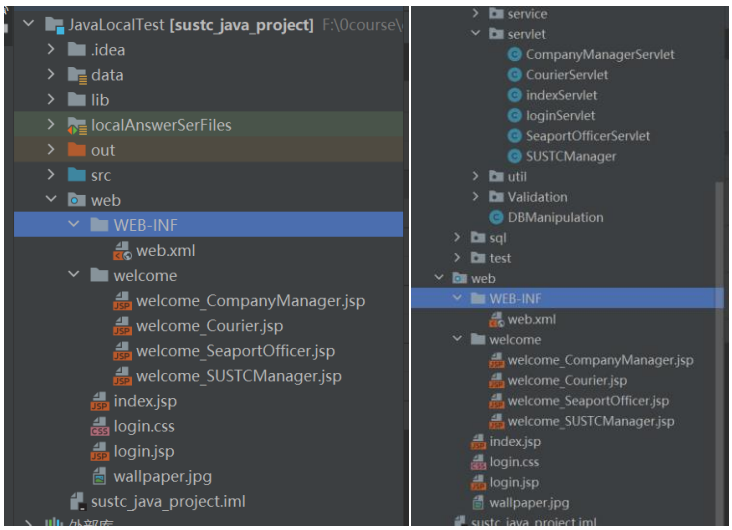
如下图所示，src 部分同 project1 主体相同，主要由持久层 model，服务层 service 以及负责与前端交互的 servlet 控制层组成。其中，model 能够直接跟数据库通信，进行数据查询或执行插入、删除、更新操作，值得注意的是，它获得的查询结果仅会以 entity 模块中定义的实体类型返回，这些实体类型的属性与数据库表严格一致，且均继承于抽象类 entity，见下图。service 层负责解构来自 servlet 层的用户需求，形成业务逻辑，同时 service 层将通过各基本数据类型以及 entity 实体类型与 model 层进行数据交换，并将必要的结果返回到 servlet 层。Servlet 层主要负责与前端进行交互，获取用户请求并进行处理，呼叫 service 层，最终将执行结果或查询结果返回给前端展示出来。

这种实现方式的优势在于，model 持久层直接与数据库交互，两者的交互逻辑较为固定、简洁，仅需实现一次，便可在多种不同的业务逻辑中进行应用。而 service 层通过实体对象与持久层进行交互，保证了整个工程的健壮性和数据的安全性。

值得注意的是，因为本次 project 有 21 个由老师定义的接口，它们实现在 DBManipulation 中，并且使用了由老师（而不是我自己的 project）定义的数据类型，如下图所示，因此 DBManipulation 会夹在 service 和 servlet 中间，形成一个额外的间接层。

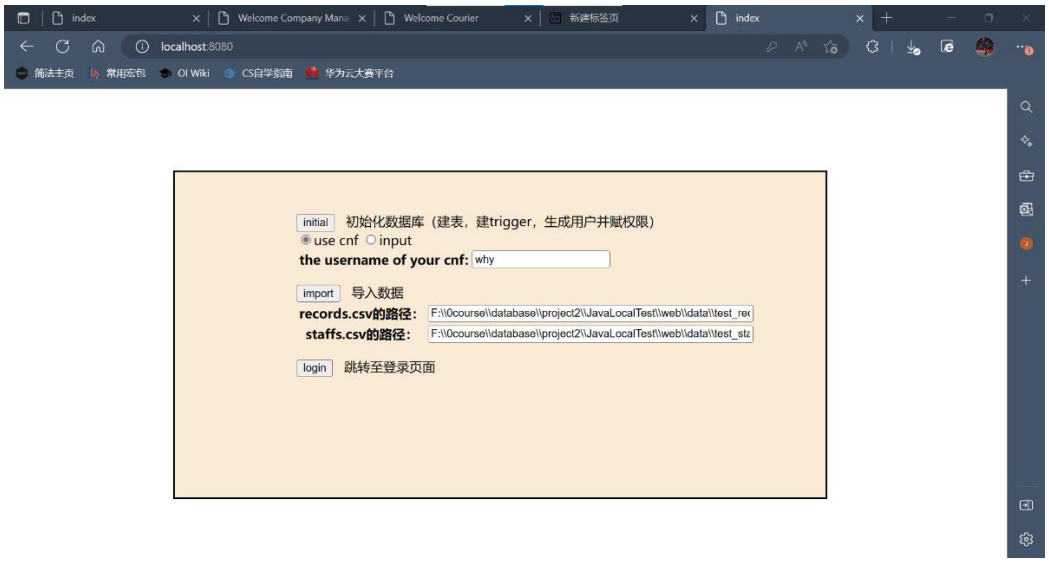


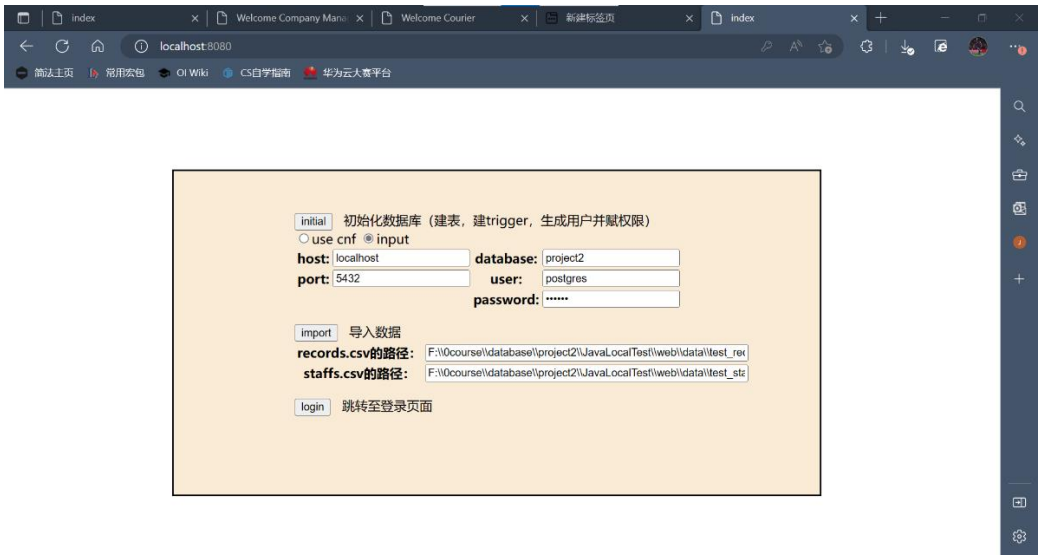
在 Tomcat 启动后，根目录为 web，各 jsp 文件均在该目录下。



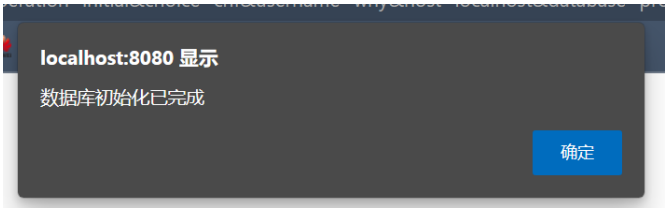
4.4 前端页面及简介如下：

1. Index.jsp：索引页面，用来数据库初始化、导入数据库。初始化主要功能是在数据库中创建表格、创建触发器以及生成之后需要用到的 user 并给其对应权限。初始化需要 root 用户权限和一个已经存在的数据库。初始化提供两种方式，分别为导入已经准备好的 cnf 文件以获得 url、用户名和密码以及手动输入的方式。导入数据需要输入两个 csv 文件的路径





数据库初始化成功时会弹窗显示：



数据导入成功时会弹窗显示：

2. 登录界面：输入用户名，密码，员工类型即可登录。最后一天通过该页面简单学习了一下 css 的相关知识。主要学习了 jsp（html）页面使用 css 文件进行修饰，包括对于背景、字体、按钮的美化以及元素位置调节的技巧。

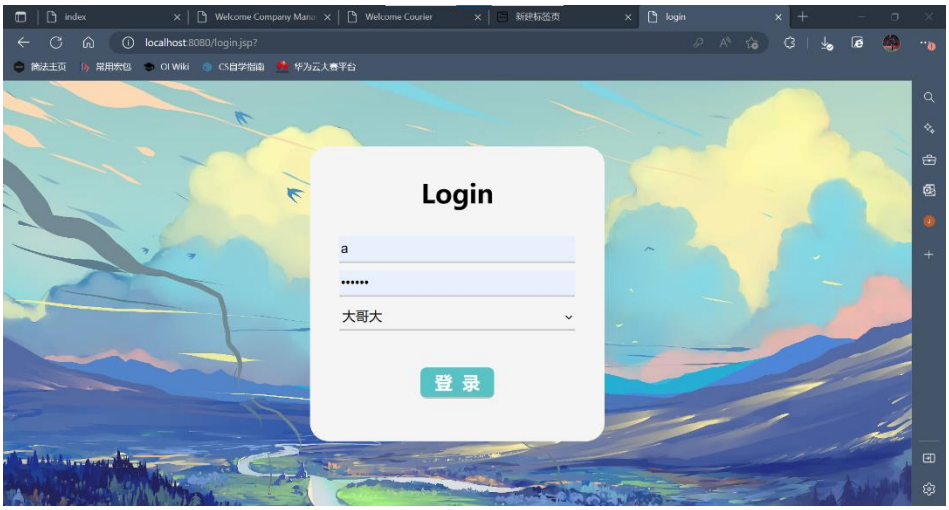
本页面代码参考及背景来源为 b 站视频：

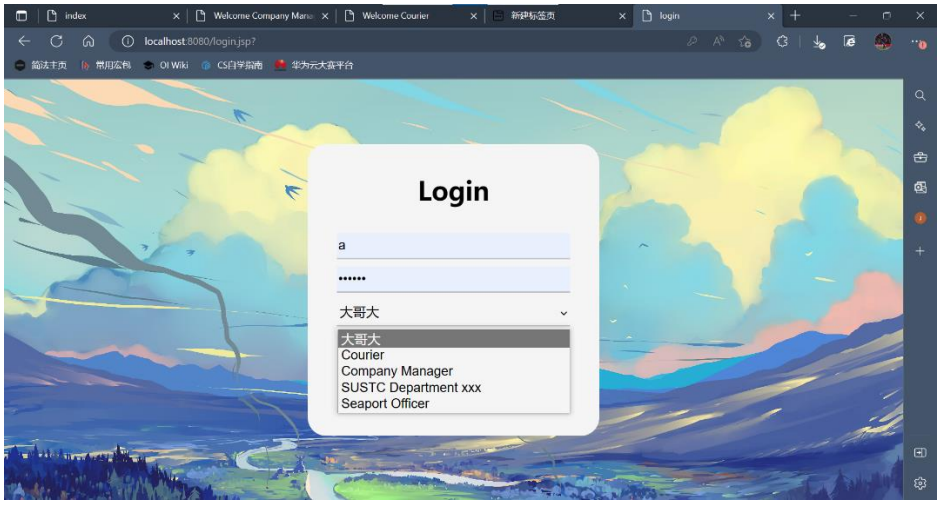
https://www.bilibili.com/video/BV1GU4y1x7iy

及对应的 github 仓库：

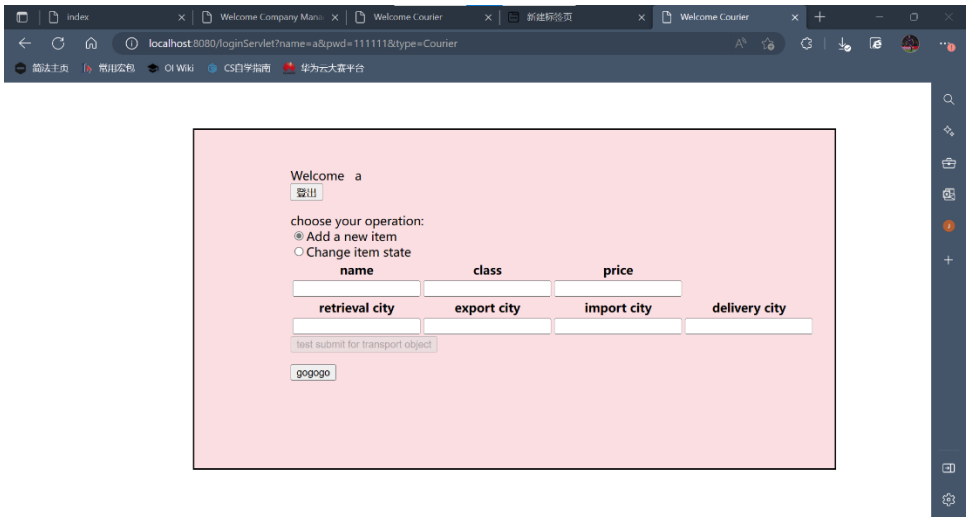
https://github.com/pokliuy/Web/tree/main/3-simpleLogin

由于时间关系，只有这一个页面画了比较多的时间进行了美化。

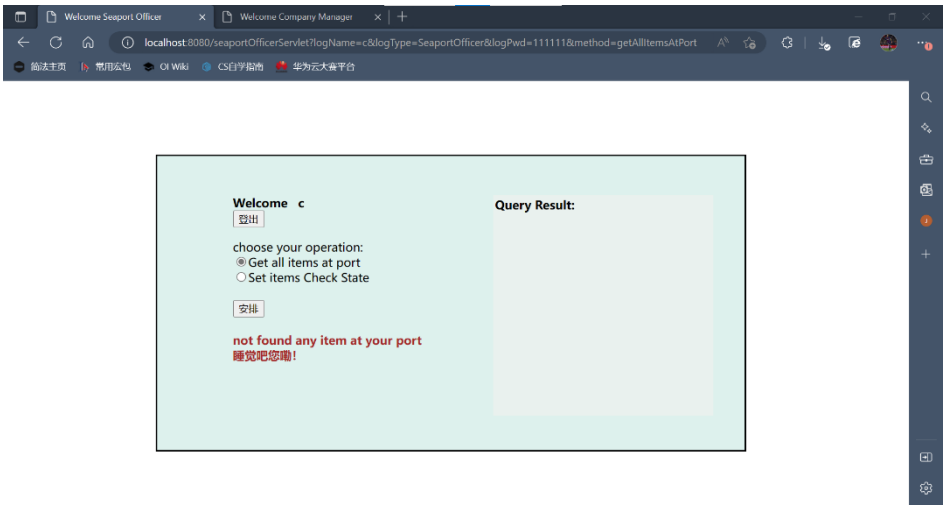




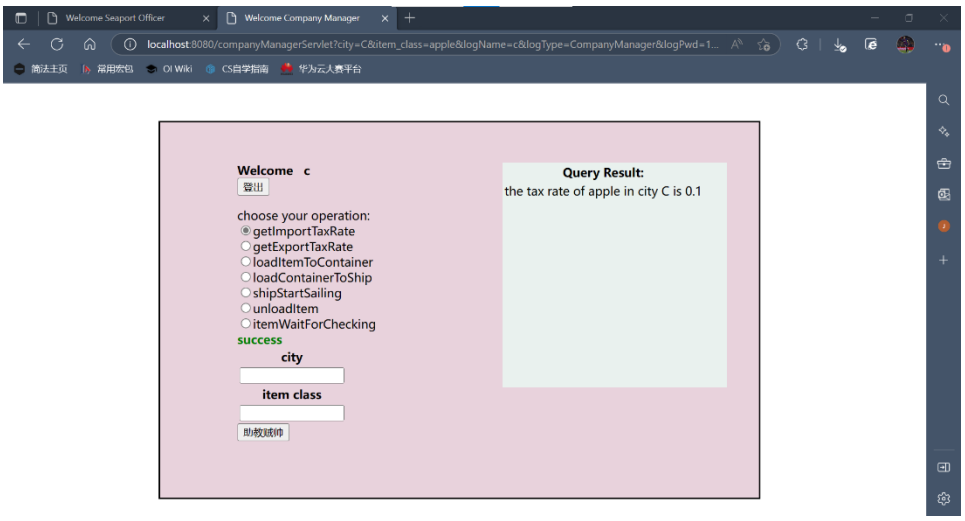
3. 进入四种类型 staff 的操作界面并执行相应操作。该界面支持“登出”操作，可以返回登录界面。下图是四种类型 staff 操作界面的示意图：



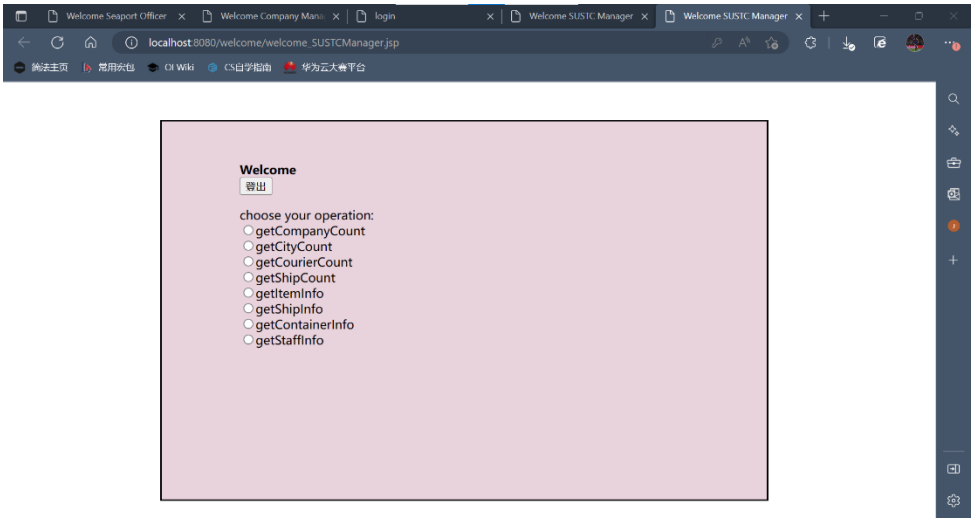
Courier



Seaport Officer



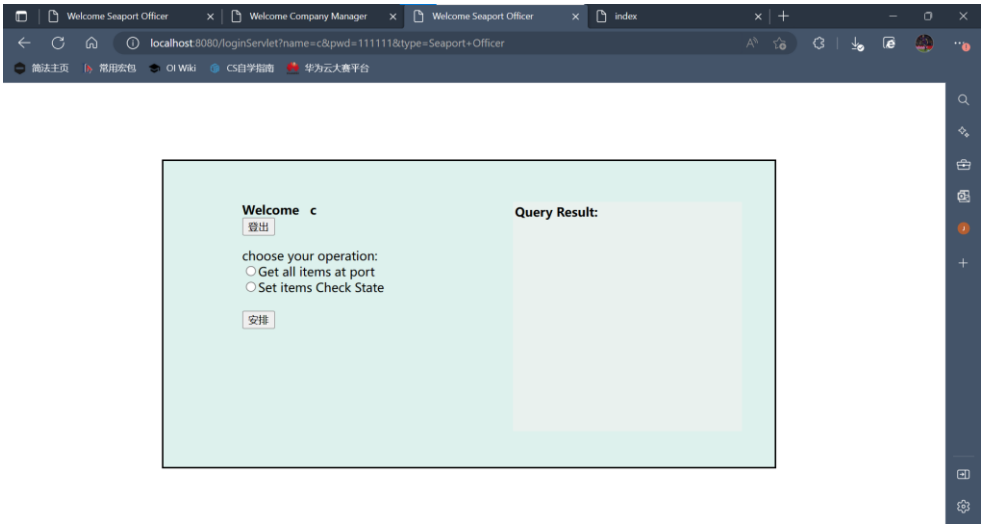
Company Manager

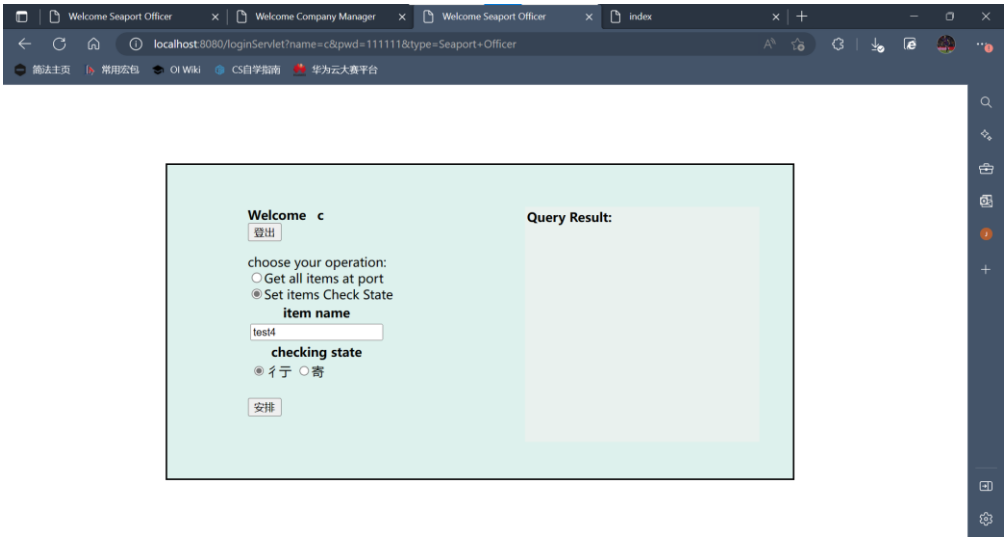


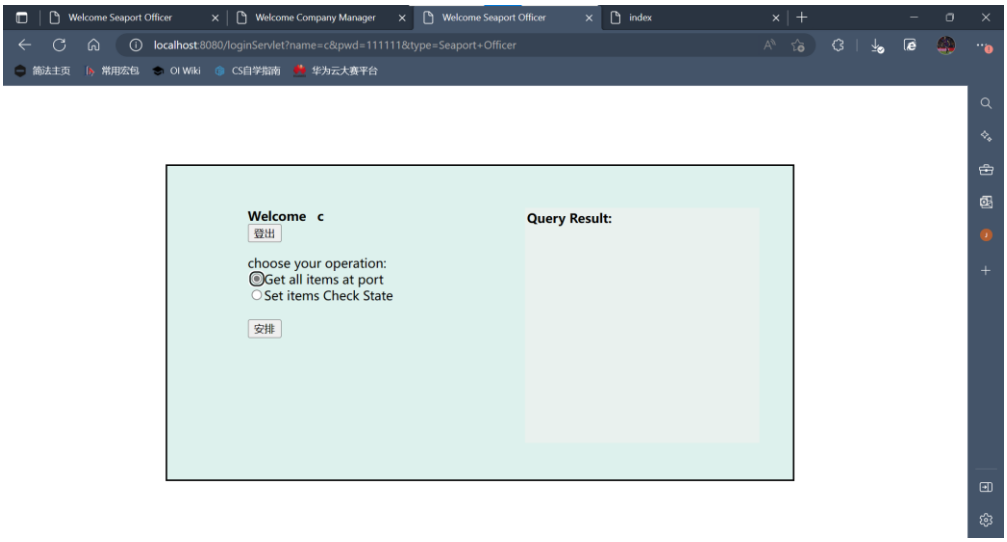
SUSTC Manager

4. 下面以 seaport officer 为例介绍操作流程（该类型包括了一个状态更改类操作 setItemCheckState 和一个查询类操作 getItemsAtPort，恰涵盖了本次 project 出现的两类操作）：

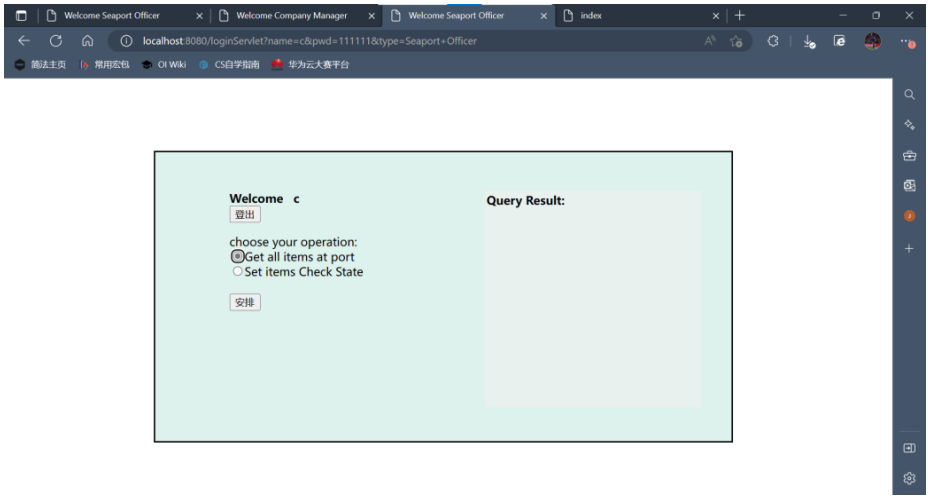
a. 初始界面，初始状态下没有进行任何选择故没有输入框也不会出现任何查询结果的信息，可以通过单击“select”组件小圆点的方式进行操作选择，在选定每个操作的同时，会浮现输入相应信息的输入框，而无关的输入框也会自动隐藏：

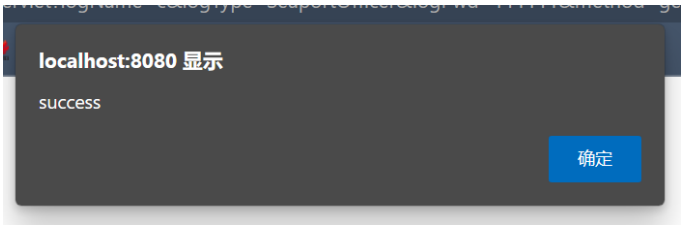




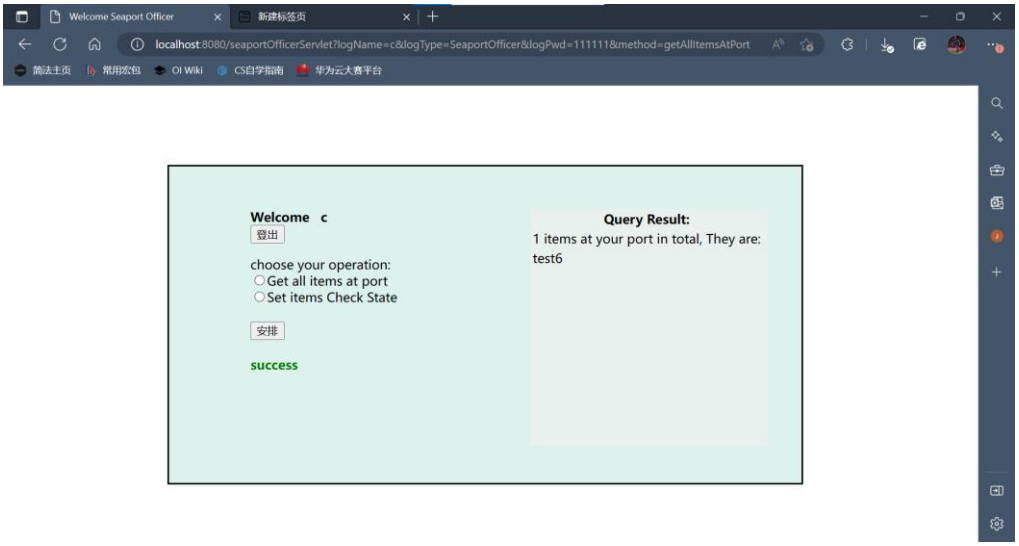


b. 查询类操作，通过输入必要参数，查询目标信息。该类操作的特点在于前端需要获取后端的返回值并展示：

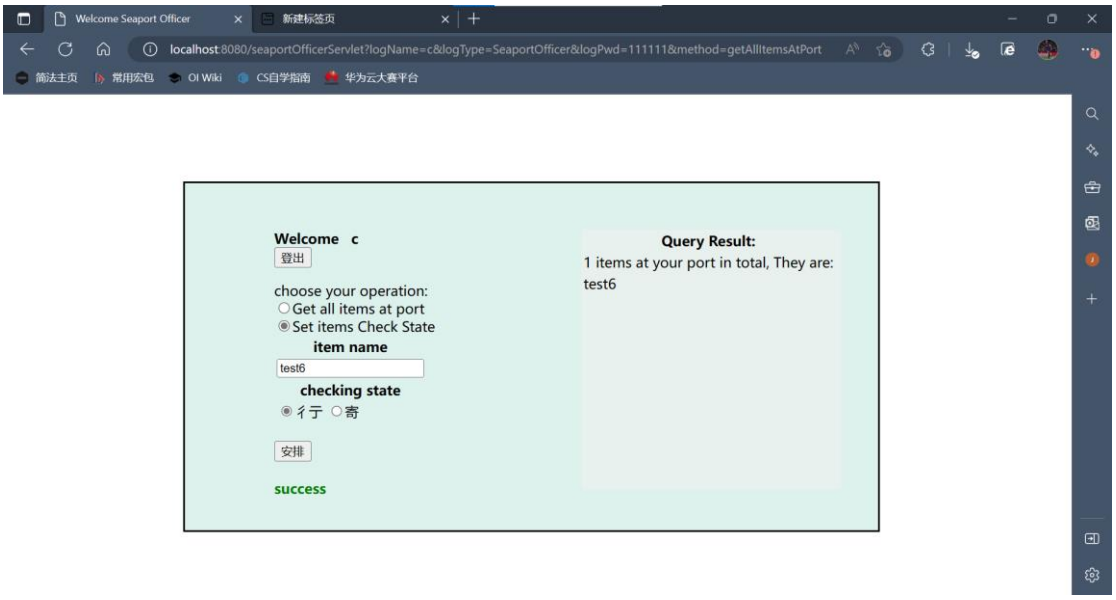




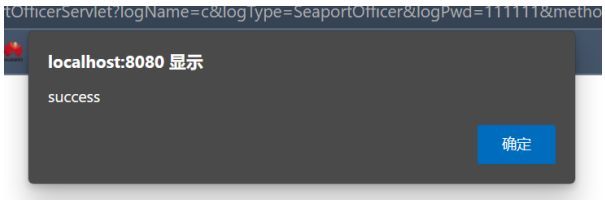
获得查询结果：



c. 状态更改类操作（如更新 delivery 状态，装货卸货、开船等）。该类操作无特别的返回值，但是仍需要反馈用户是否操作成功：

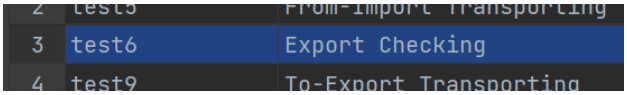


点击“安排”，弹窗显示：

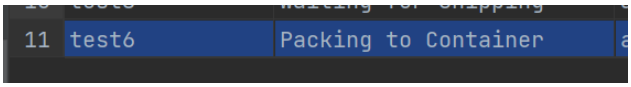


即操作成功

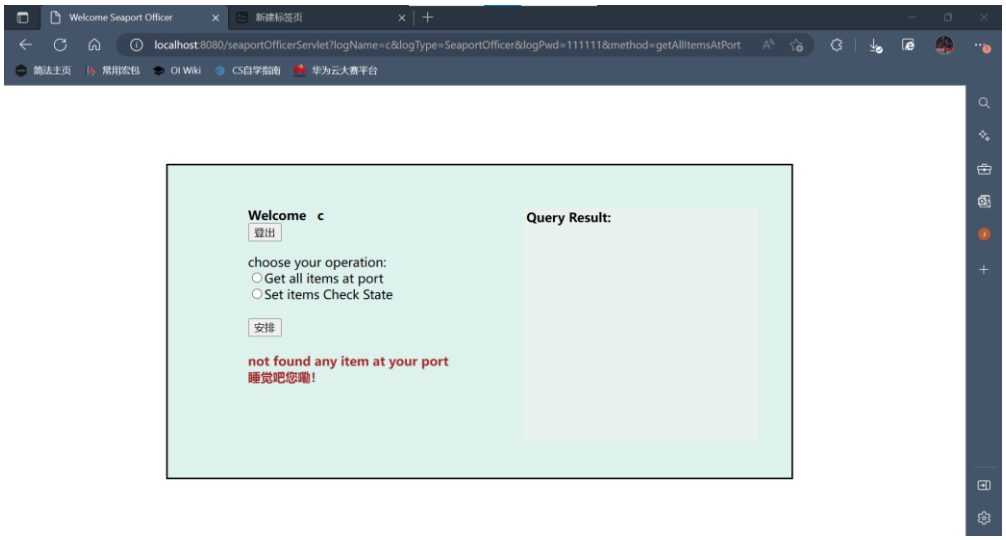
数据库中可观察到，操作前，test6 状态为“Export Checking”：



操作后，状态变为“Packing to Container”：



若再次查询该港口的物品，会发现：



该港口已经没有待检查的物品了。

5. Summary

我们以本次 project 为契机，学习了用 java 结合 tomcat，jsp，css 等工具去实现前后端的交互以及前端的美工，在开阔眼界的同时，给本次 project 画上了圆满的句号。相信在之后各门计算机系课程的项目中，我们能够利用本次 project 所学到的知识，设计出更漂亮的结构，创造出更独特、更具美感的展现方式。