

**CS307 Principles of Database Systems**

**Project 2 Report**

**Database of SUSTC**

11911612 Haoyu Wang

12112011 YiYu Liu

class 1 group 1

December 23, 2022

Content

[Contributions 3](#_Toc129987723)

[Task 1: Database Design 4](#_Toc129987724)

[ER Diagram and Database Design of project 2: 4](#_Toc129987725)

[Visualization of DataGrip 5](#_Toc129987726)

[Descriptions on Tables 5](#_Toc129987727)

[User Authorization Management 6](#_Toc129987728)

[Trigger 9](#_Toc129987729)

[Task 2: Basic APIs implementation And Analyses 13](#_Toc129987730)

[Basic APIs implementation structures 13](#_Toc129987731)

[Implementation of DAOs 14](#_Toc129987732)

[Optimization of APIs on running time 16](#_Toc129987733)

[Task 3: Advanced APIs And Frontend Design 18](#_Toc129987734)

[概述 18](#_Toc129987735)

[操作逻辑流程： 19](#_Toc129987736)

[Java 项目结构： 19](#_Toc129987737)

[前端页面及简介： 20](#_Toc129987738)

[Summary 28](#_Toc129987739)

# 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

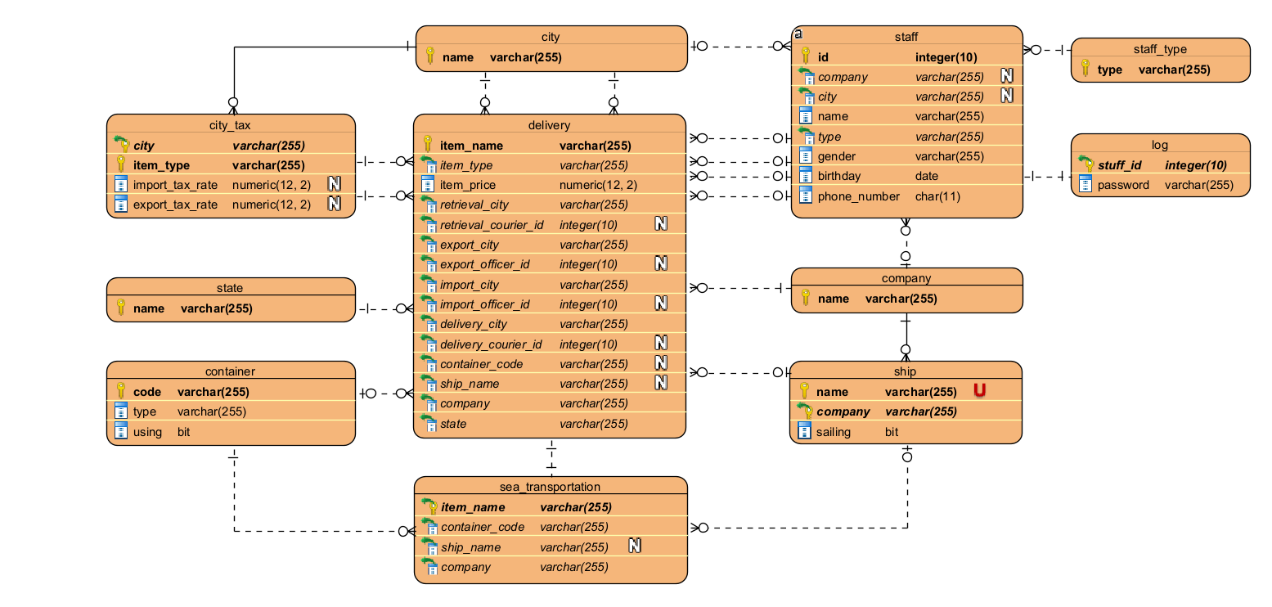
**Yiyu Liu:**

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

# Task 1: Database Design

## ER Diagram and Database Design of project 2:

using Visual Paradigm



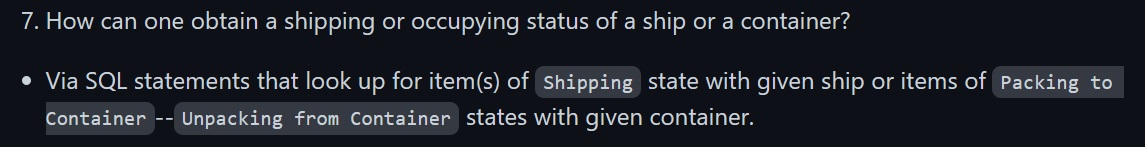
(for each attribute, sign N means it can be null, sign U means it is unique)

**The Design of Database:**

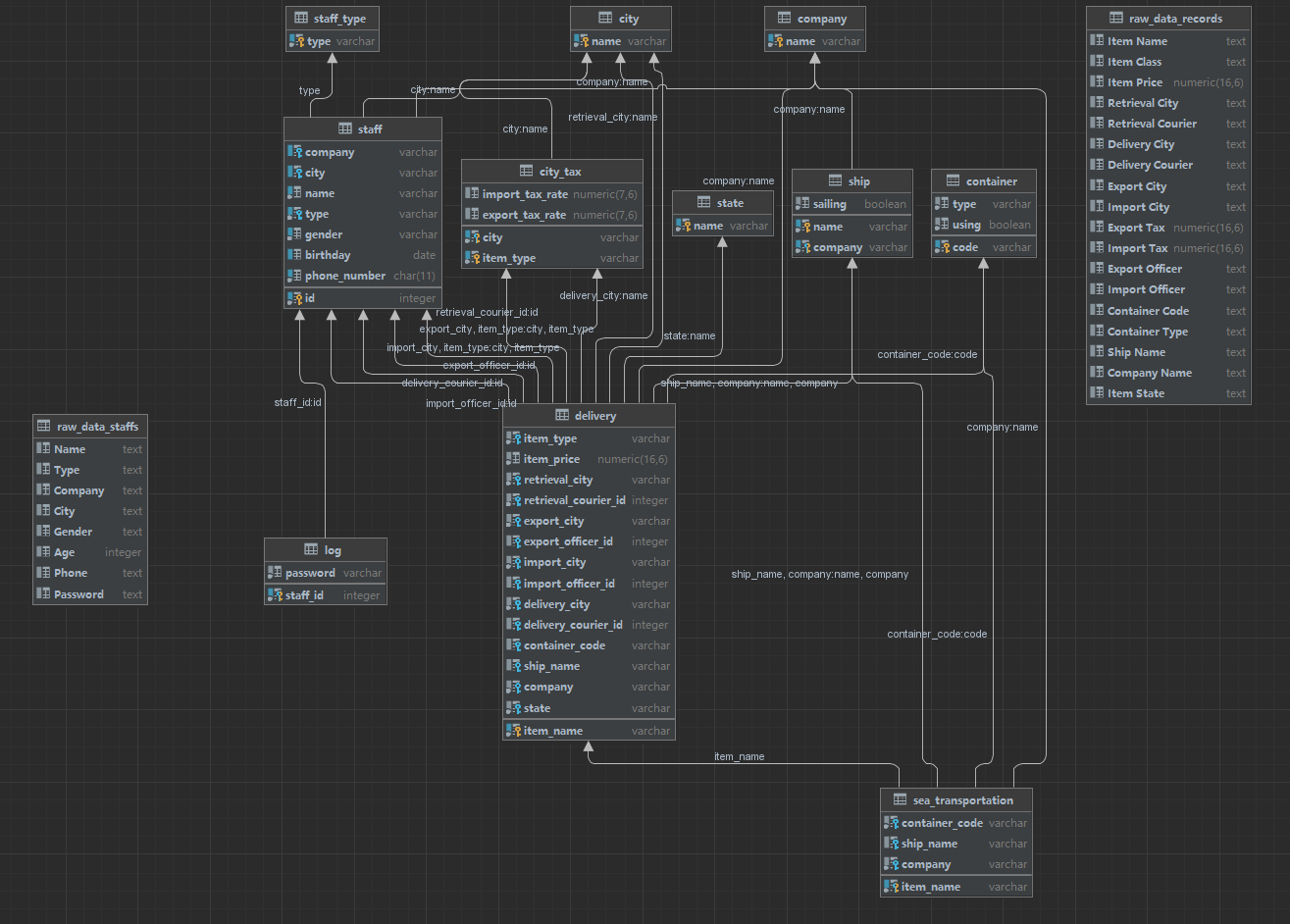
1. Two Primary Key in table “city tax” and “ship”: confirm the consistency of database
2. The field “using” of table “container” and the field “sailing” of table ship: they are extra fields from project 1
3. Table “sea transportation”: This is a dynamic table, which records item and its current container 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。

1. 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。



## Visualization of DataGrip



## Descriptions on Tables

1. **Basic information of tables**
2. “raw\_data\_records”: original data from records.csv.
3. “raw\_data\_staffs”: original data from staffs.csv.
4. “city”: city names.
5. “city\_tax”: for port city, it records export and import tax rates corresponds to item class
6. “state”: state of each record.
7. “container”: code, type, whether it is using.
8. “ship”: name, company, whether it is sailing.
9. “company”: company names.
10. “staff”: id, company, city, name, staff types, gender, birthday, phone number.
11. “staff\_type”: all staff types in “SUSTC Department Manager”, “Courier”,

“Company Manager”, “Seaport Officer”.

1. “log”: staffs with their passwords.
2. “delivery”: information of each record.
3. “sea\_transportation”: information of records that states are during “Packing to

Container” and “Unpacking from Container”.

1. **Specifications on tables**
2. 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。

1. Table “sea transportation” is a dynamic table. We design it in order to manage

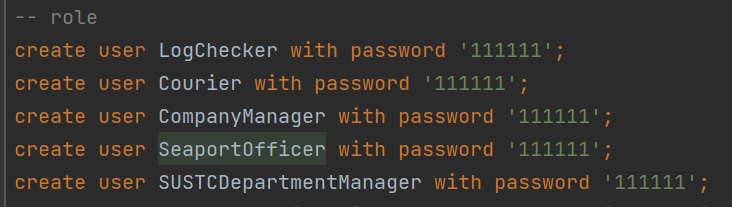
state transferring of delivery records, which has been introduced in The Design of Database.

1. We use id to specify different staffs. However, in this project, each staff has a

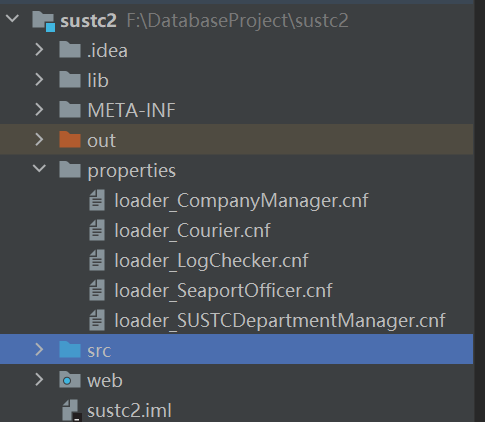
unique name. Hence in sql queries we may only use the condition of identical names to confirm sameness.

## User Authorization Management

We have five users in total:



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:

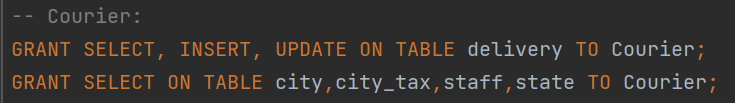
1. **Log Checker：**

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



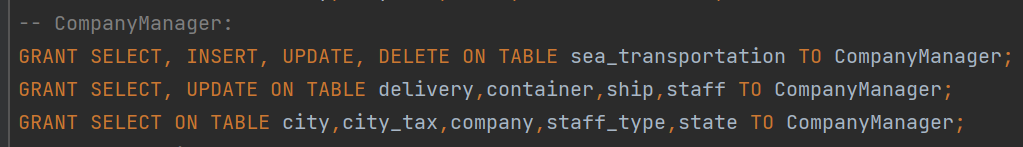
1. **Courier**

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



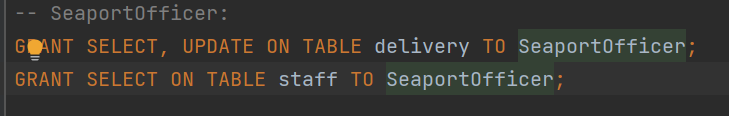
1. **Company Manager**

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.



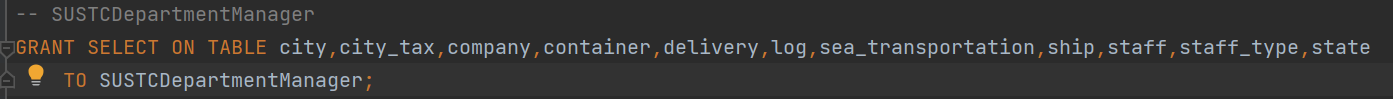
1. **Seaport Officer**

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



1. **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.

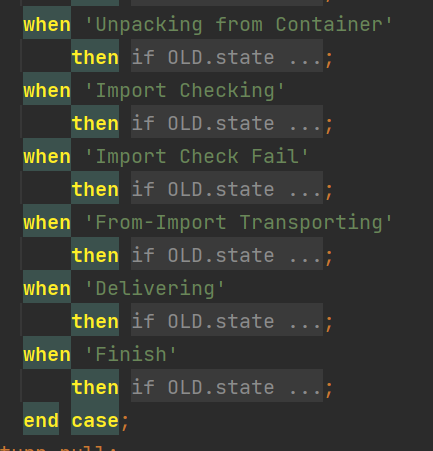
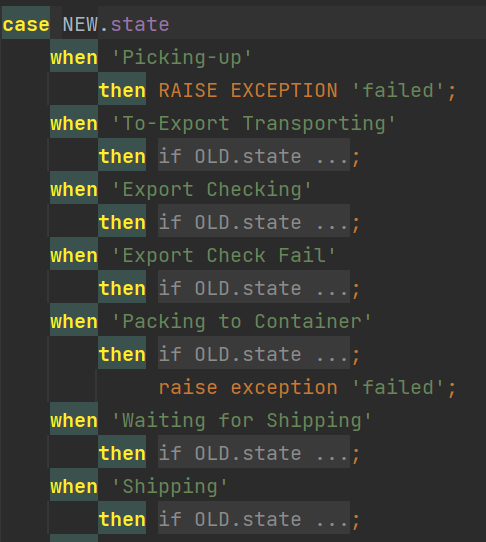


## Trigger

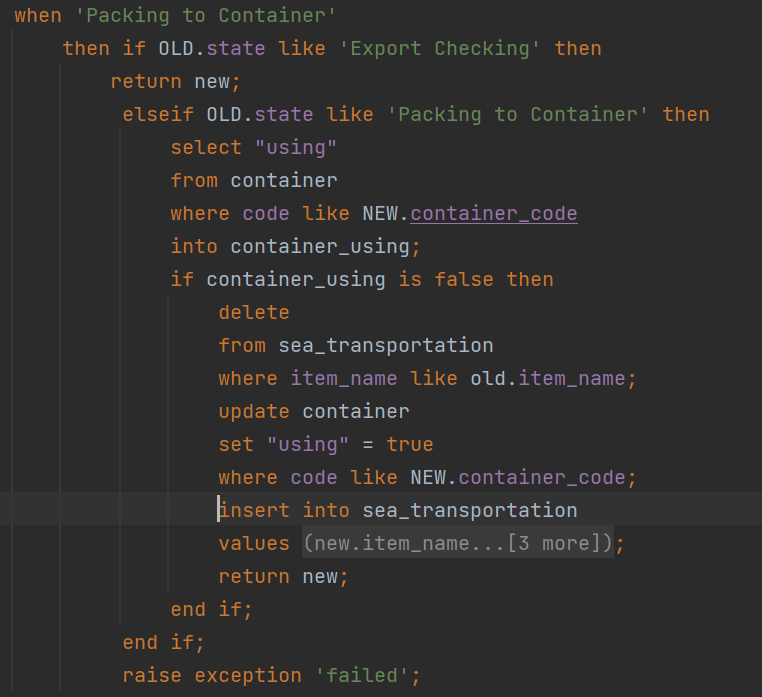
We have four triggers in total. For convenience, we name each trigger by its function. All of these triggers have been set as “before operation” and “for each row”. Detail:

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

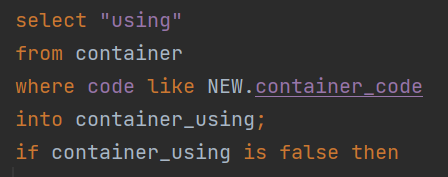




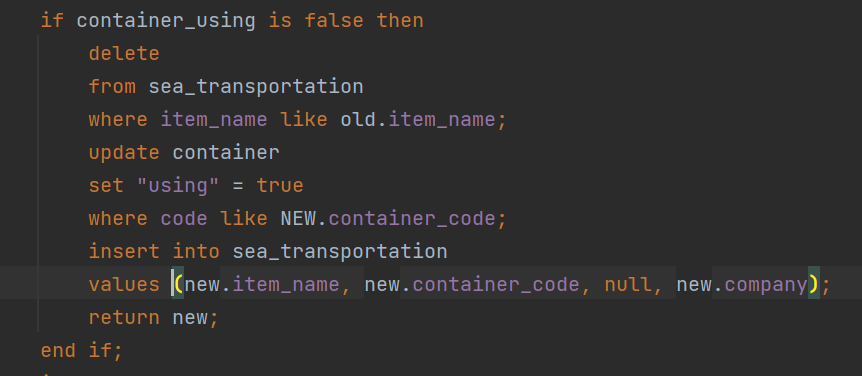
* 1. When item state is “Packing to Container”:



* + 1. Check if the container using field is false;



* + 1. Then we need to do 3 operations, delete, update, insert:

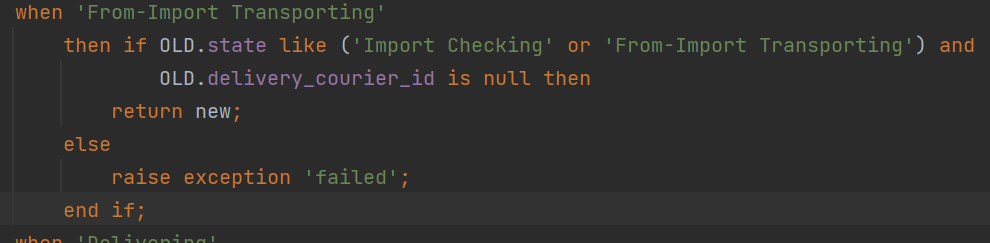


Firstly, delete from table “sea transportation”. It is used to deal with this item have been packed into another container before. Secondly, set container field using is true. Thirdly, insert a new record into table “sea transportation”

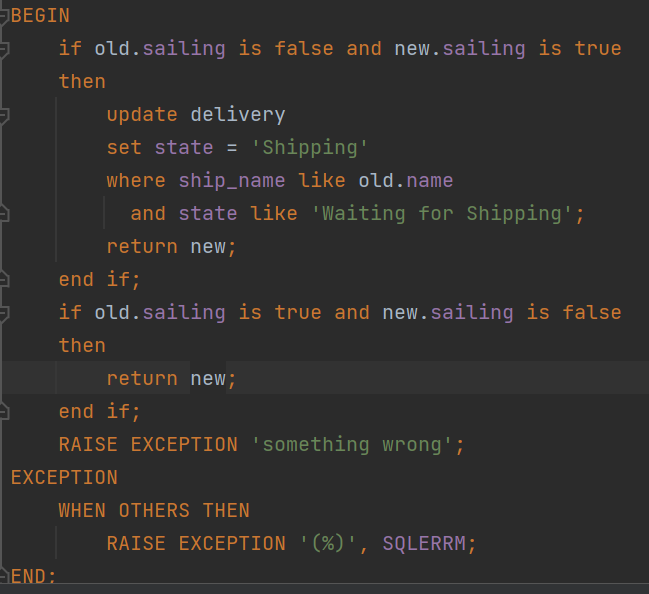
当物品装载进一个集装箱时，我们会在表“sea transportation”中新建一个记录同时更新集装箱状态。但是这个物品有可能发生从集装箱A换到集装箱B的情况。这个时候，我们就需要先删除之前在表“sea transportation”中的记录，再重新插进去一个新的记录，而集装箱A的状态将通过触发器“delete trigger”去更新。

* 1. 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”.



1. **update ship**

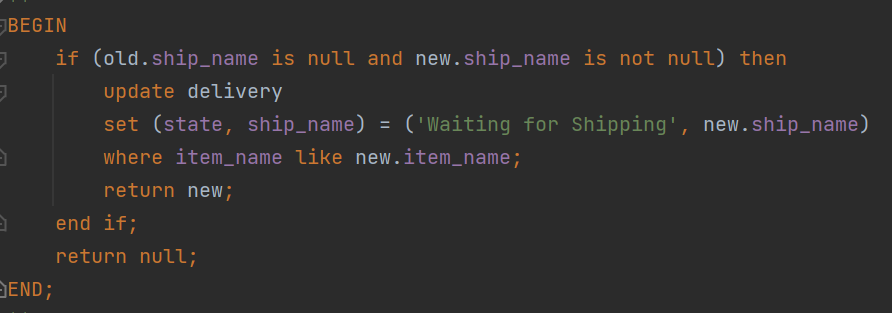


* 1. The important point of field sailing in table “ship” is that when a company manager ask a ship to sail, all of the items in this ship should change their state from “Waiting for Shipping” into “Shipping”.

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”

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

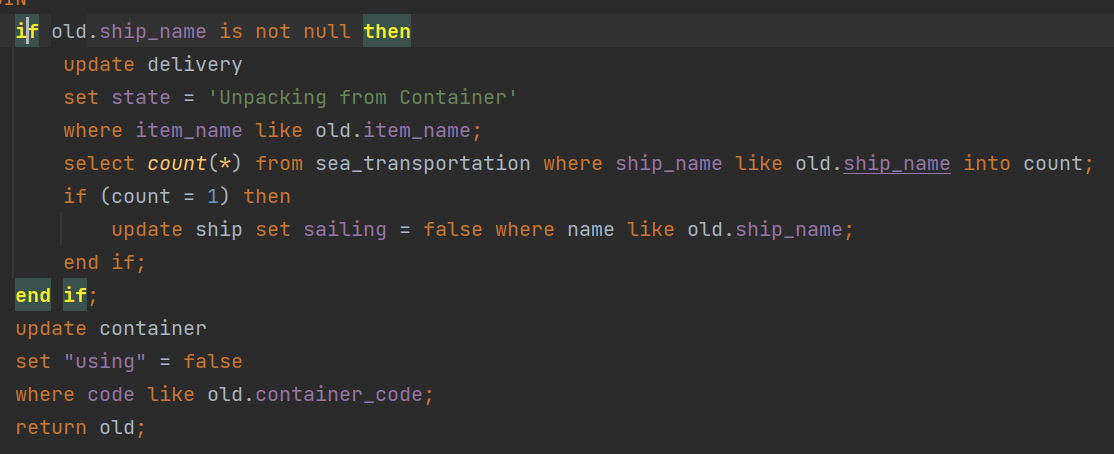
1. **update sea transportation**



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。

1. **delete sea transportation**



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

* 1. an item is moved from container A into container B, whose state must be “Packing to Container”
  2. 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。

# Task 2: Basic APIs implementation And Analyses

## Basic APIs implementation structures

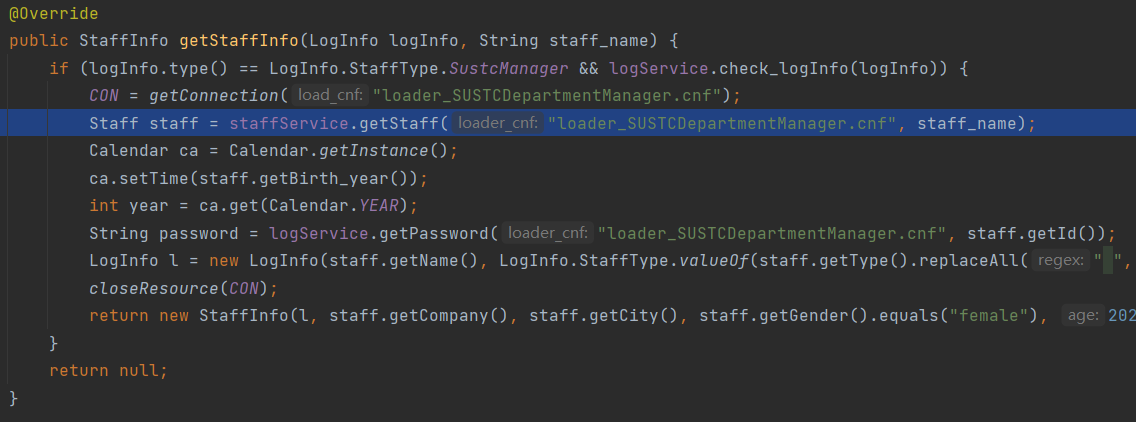
In our project, all basic APIs are implemented at bottom through DAO(Database Access Objects). When a query is put forward (specifically, an interface method of API is called), after checking user’s authorization, our system calls the service class needed, which then calls its subordinate DAO class to access the table needed in the database, thus to achieve corresponding instructions.

All of these interface methods are placed in class DBManipulation.

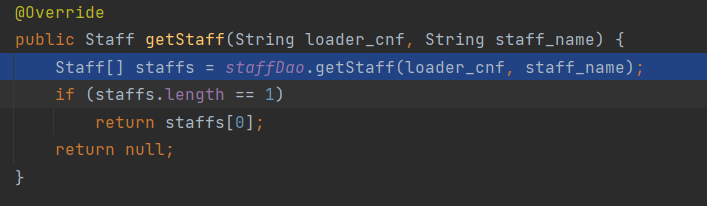
在我们的项目中，所有的基础API在底层都是通过DAO（数据库访问对象）实现的。当提出指令时（即调用API的接口方法），我们的系统在核查用户权限后，将调用所需的服务器类，服务类再调用其下属的DAO类来访问数据库中所需的表，以实现相应的指令。

所有这些接口方法都在类DBManipulation中。

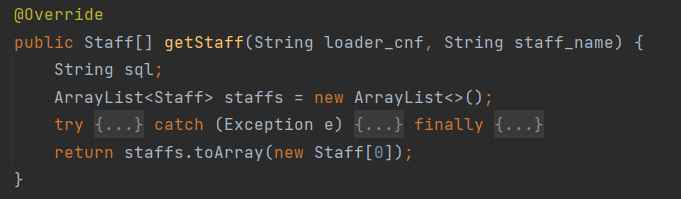












## Implementation of DAOs

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.



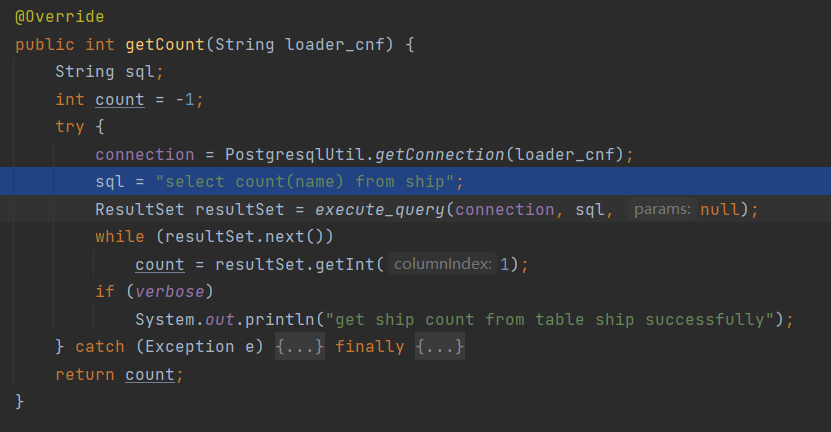
For methods of different staffs, the specific terms needed to fill in the statements are passed in as parameters. The user info is also among that.

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:

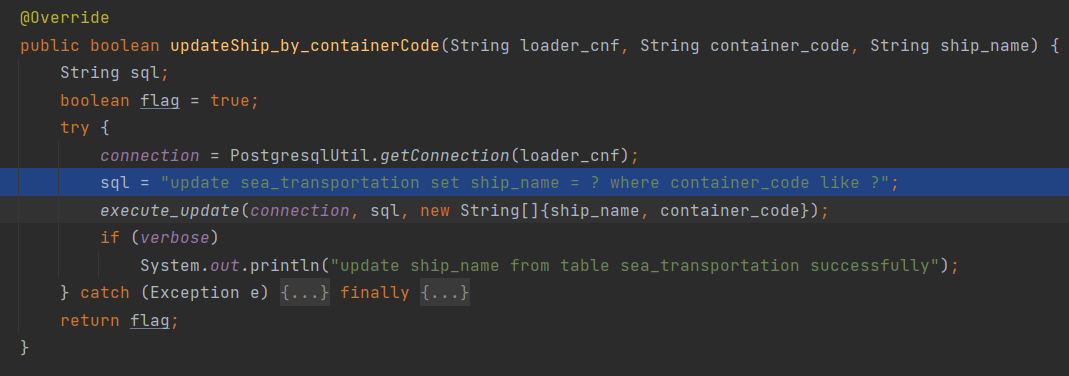












## Optimization of APIs on running time

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.

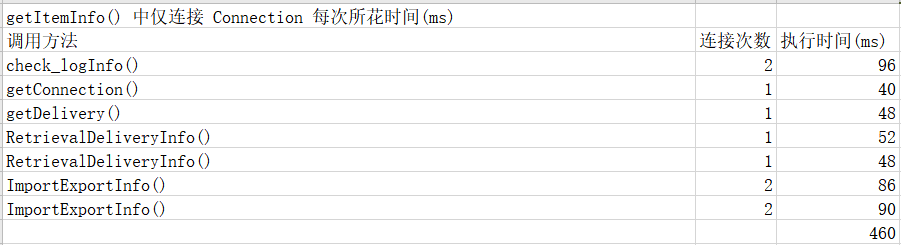
在测试中我们发现，主要的时间成本是连接数据库，每次连接将花费约40 ms。在原始设计中，为保证每次对数据库操作的独立性，我们在一个DBManipulation内可能进行多次数据库连接。下面以getItemInfo()方法进行举例。

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。

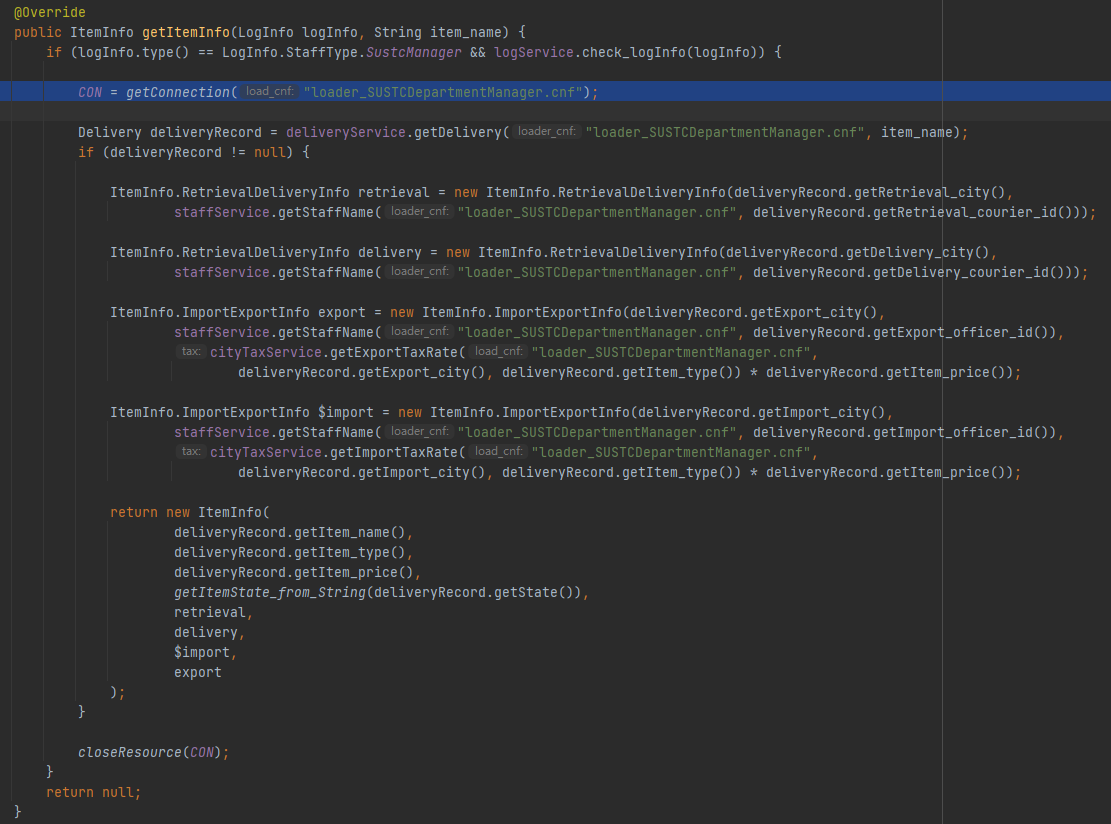
使用DBManipulation类中的成员连接或使用全局连接可以解决这个问题。因为DBManipulation中的每个接口方法都依赖于单个特定用户，所以我们只需要获得一次连接。这样花费的总时间为200-300 ms。

节约了约75%的时间消耗。

具体**数据**如图所示：



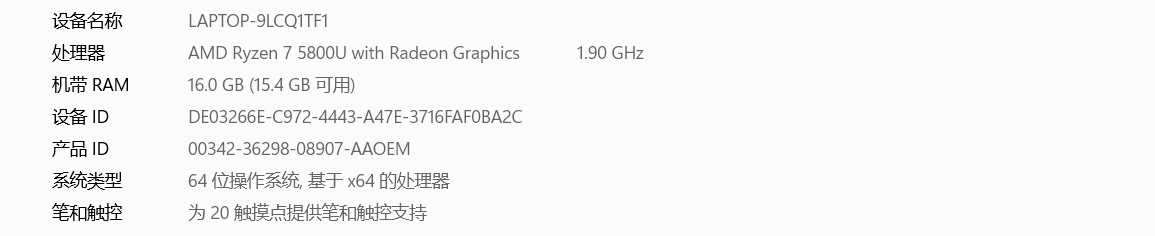
如图：在该方法开始部分使用全局变量CON连接数据库，下面各方法在具体的实现过程中，均直接使用了该全局变量：



**附：测试环境**

系统环境：Windows

其他配置如下图所示：



# Task 3: Advanced APIs And Frontend Design

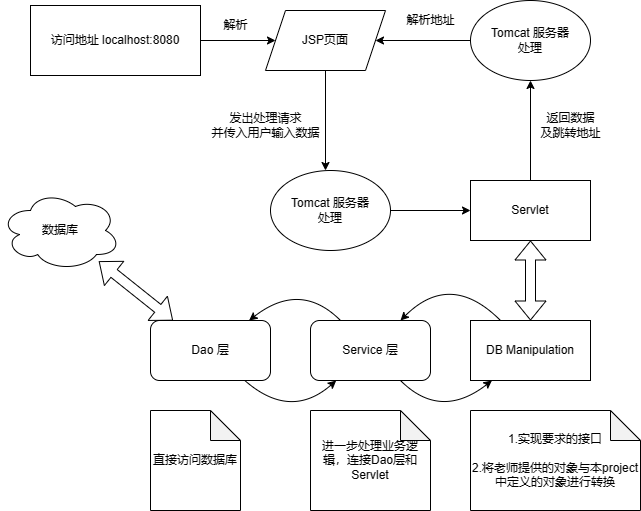
*注：为便于表述及阅读，本部分完全使用中文书写*

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：

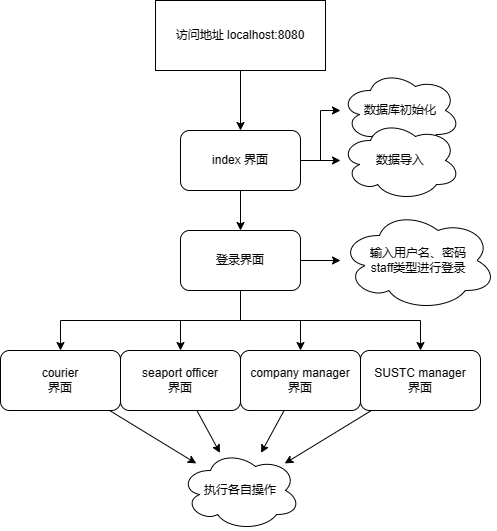
## 概述

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）



## 操作逻辑流程：



## Java 项目结构：

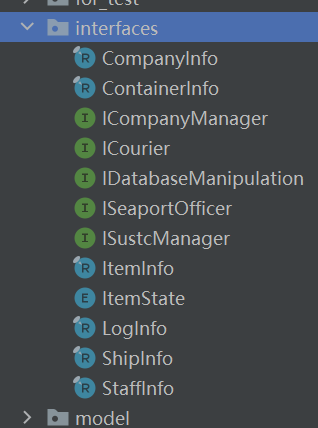
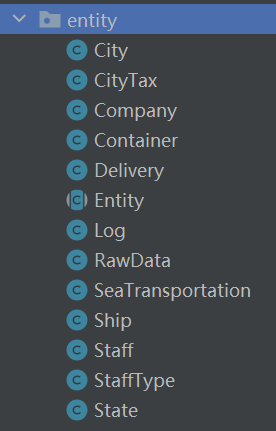
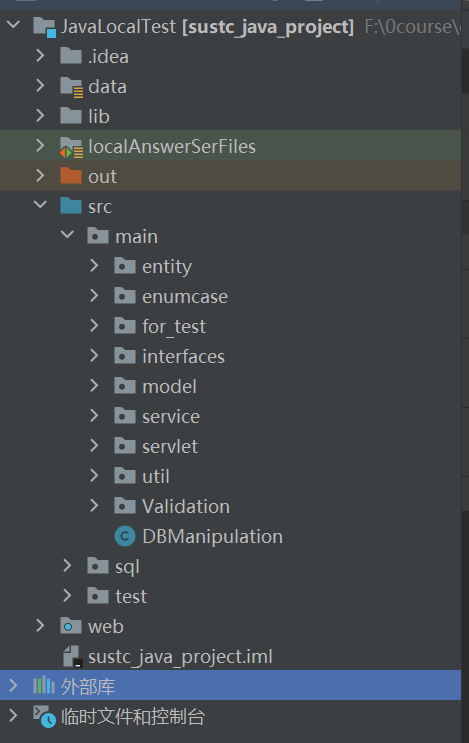
如下图所示，src部分同project1主体相同，主要由持久层model，服务层service以及负责与前端交互的servlet控制层组成。其中，model能够直接跟数据库通信，进行数据查询或执行插入、删除、更新操作，值得注意的是，它获得的查询结果仅会以entity模块中定义的实体类型返回，这些实体类型的属性与数据库表严格一致，且均继承于抽象类entity，见下图。

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。

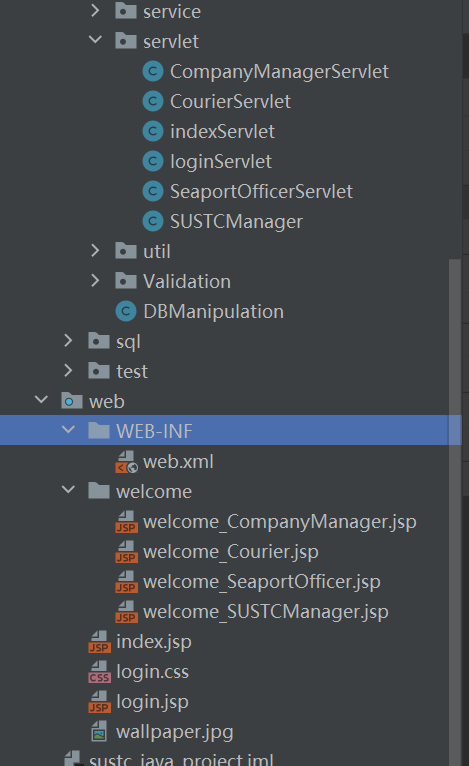
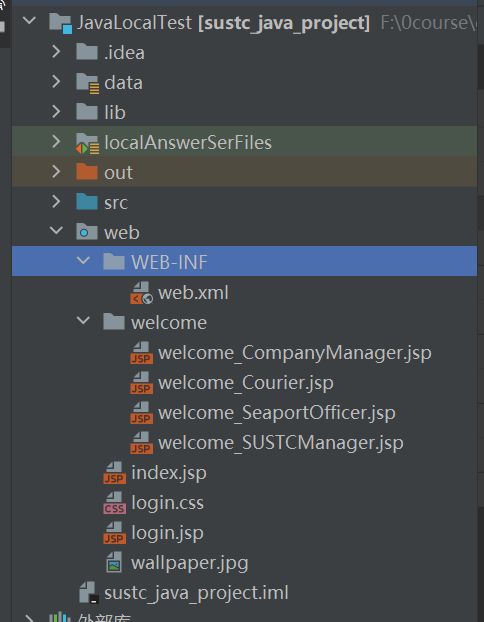
Servlet层主要负责与前端进行交互，获取用户请求并进行处理，呼叫service层，最终将执行结果或查询结果返回给前端展示出来。

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。

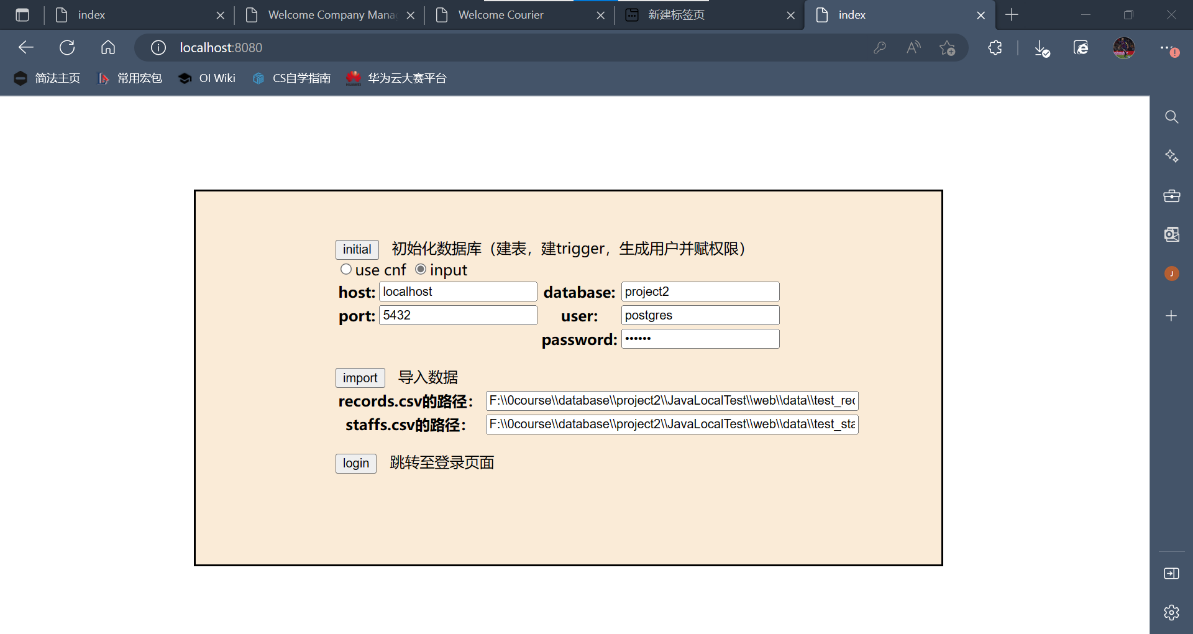
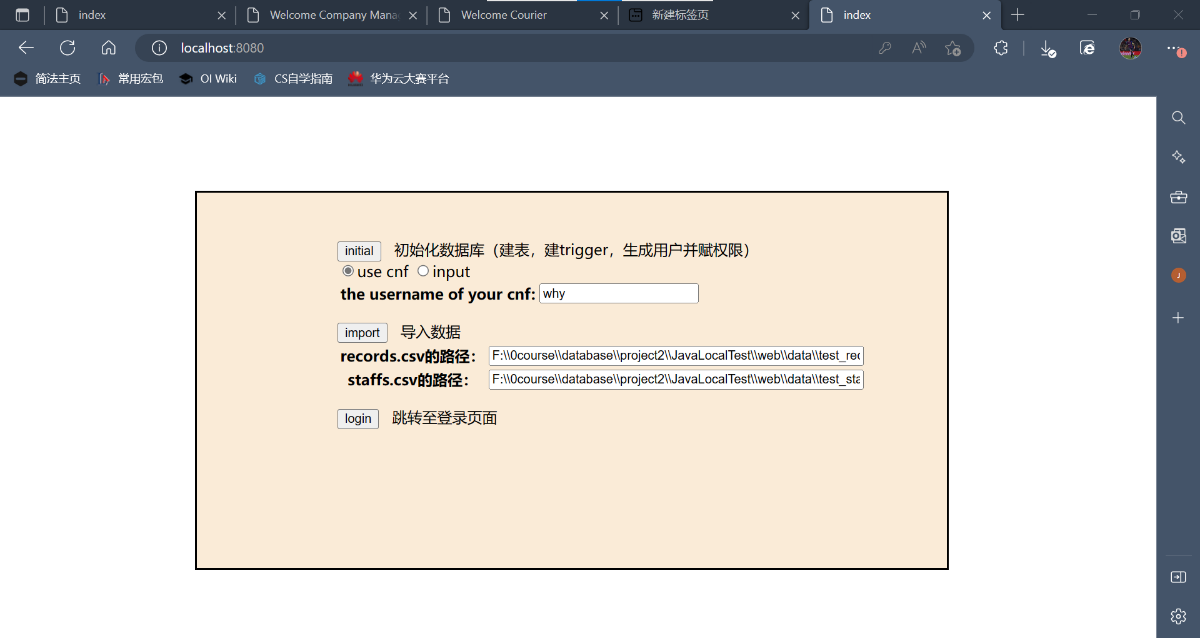


在Tomcat启动后，根目录为web，各jsp文件均在该目录下。

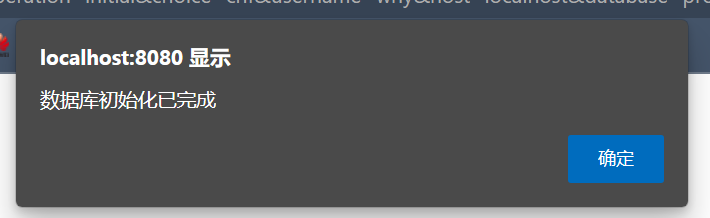


## 前端页面及简介：

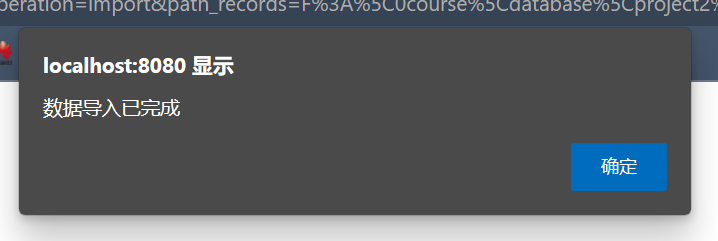
1. Index.jsp：索引页面，用来数据库初始化、导入数据库。初始化主要功能是在数据库中创建表格、创建触发器以及生成之后需要用到的user并给其对应权限。初始化需要root用户权限和一个已经存在的数据库。初始化提供两种方式，分别为导入已经准备好的cnf文件以获得url、用户名和密码以及手动输入的方式。导入数据需要输入两个csv文件的路径



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：



1. 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。

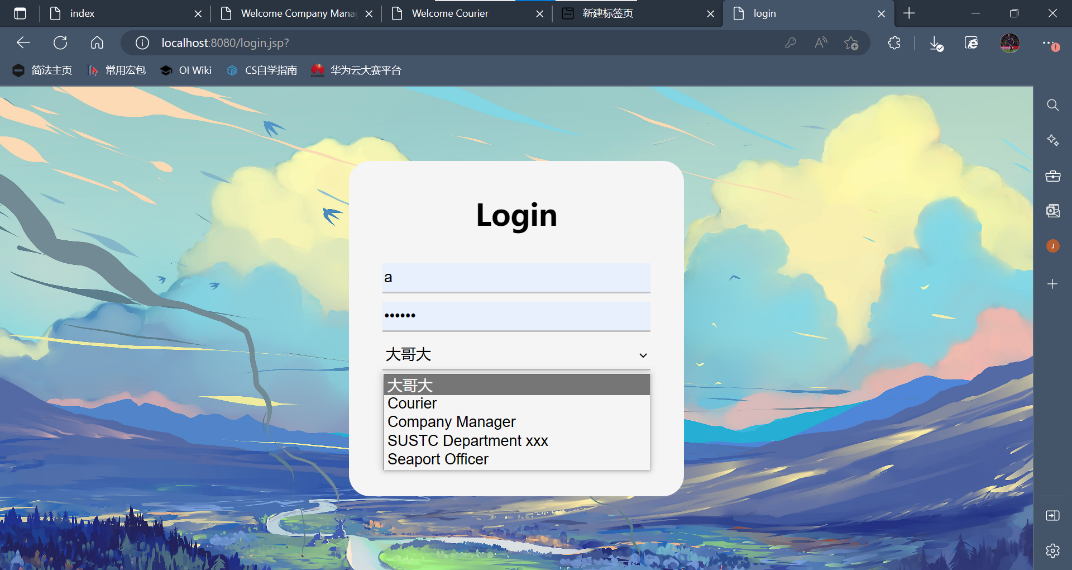
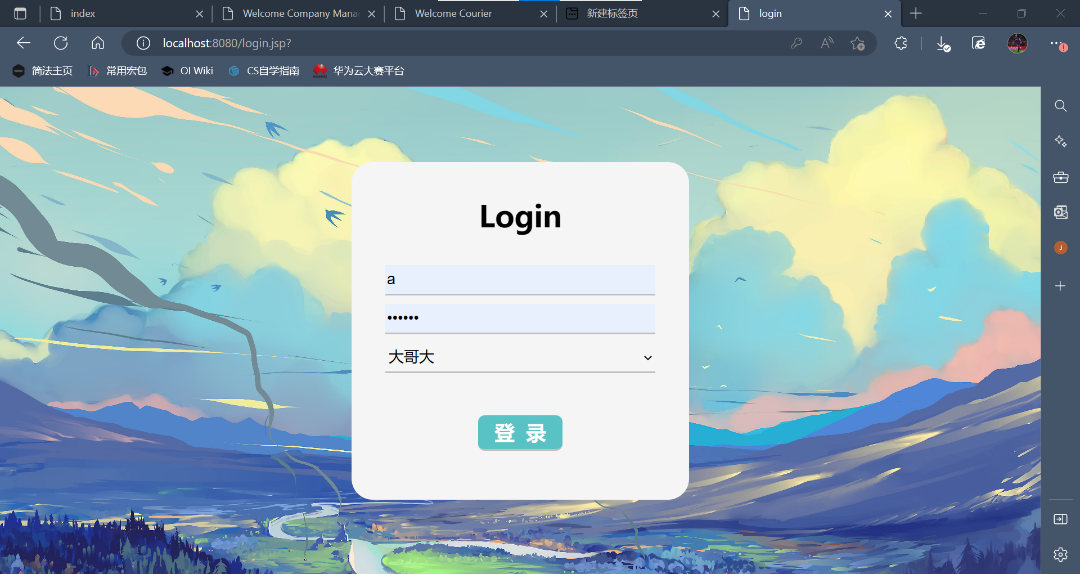
本页面代码参考及背景来源为b站视频：

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

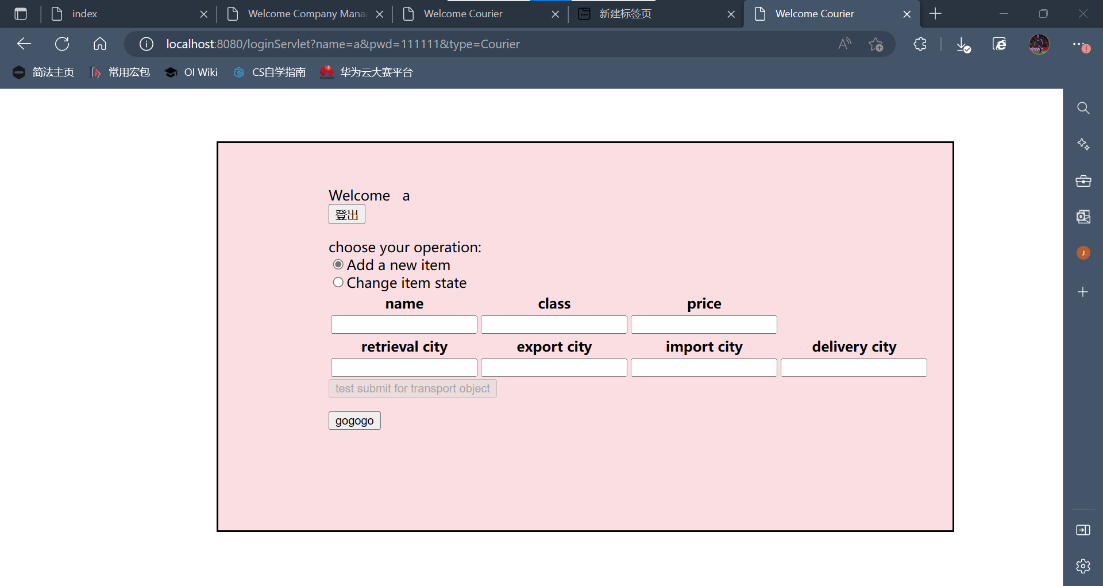
及对应的github仓库：

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

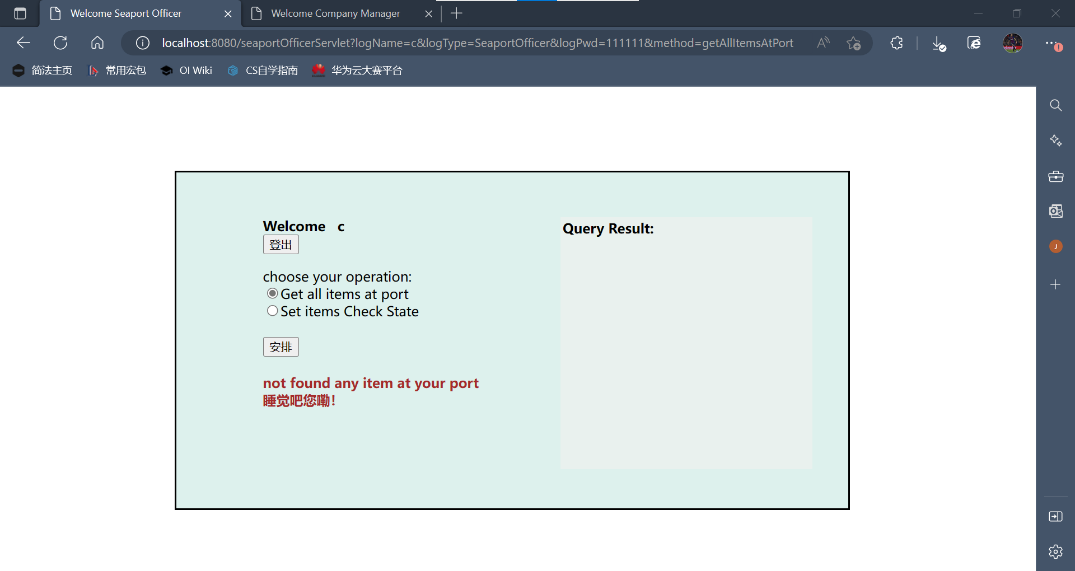
由于时间关系，只有这一个页面画了比较多的时间进行了美化。



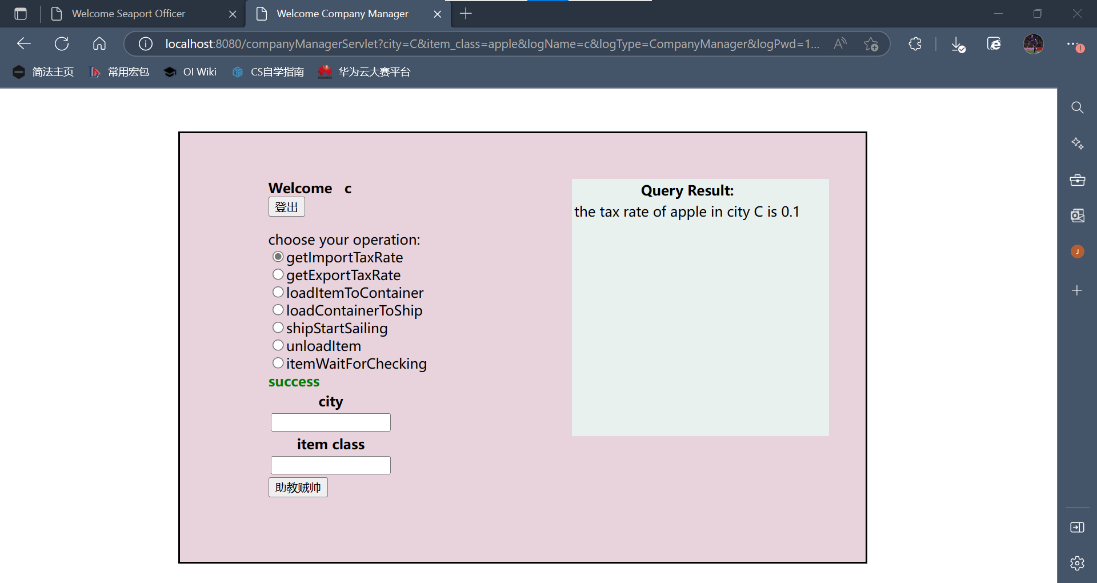
1. 进入四种类型staff的操作界面并执行相应操作。该界面支持“登出”操作，可以返回登录界面。下图是四种类型staff操作界面的示意图：



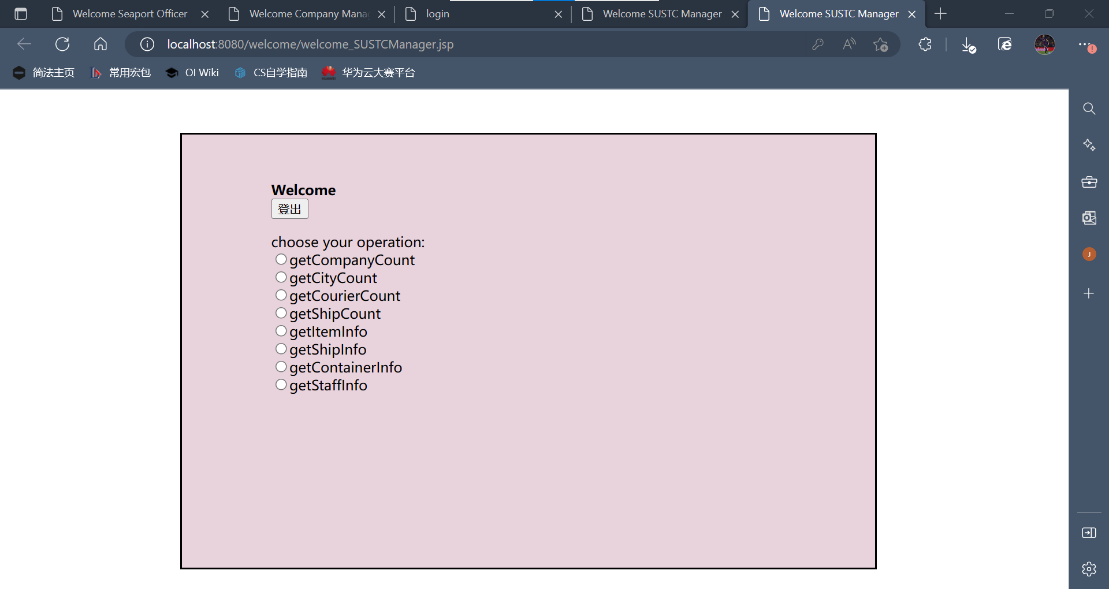
Courier



Seaport Officer

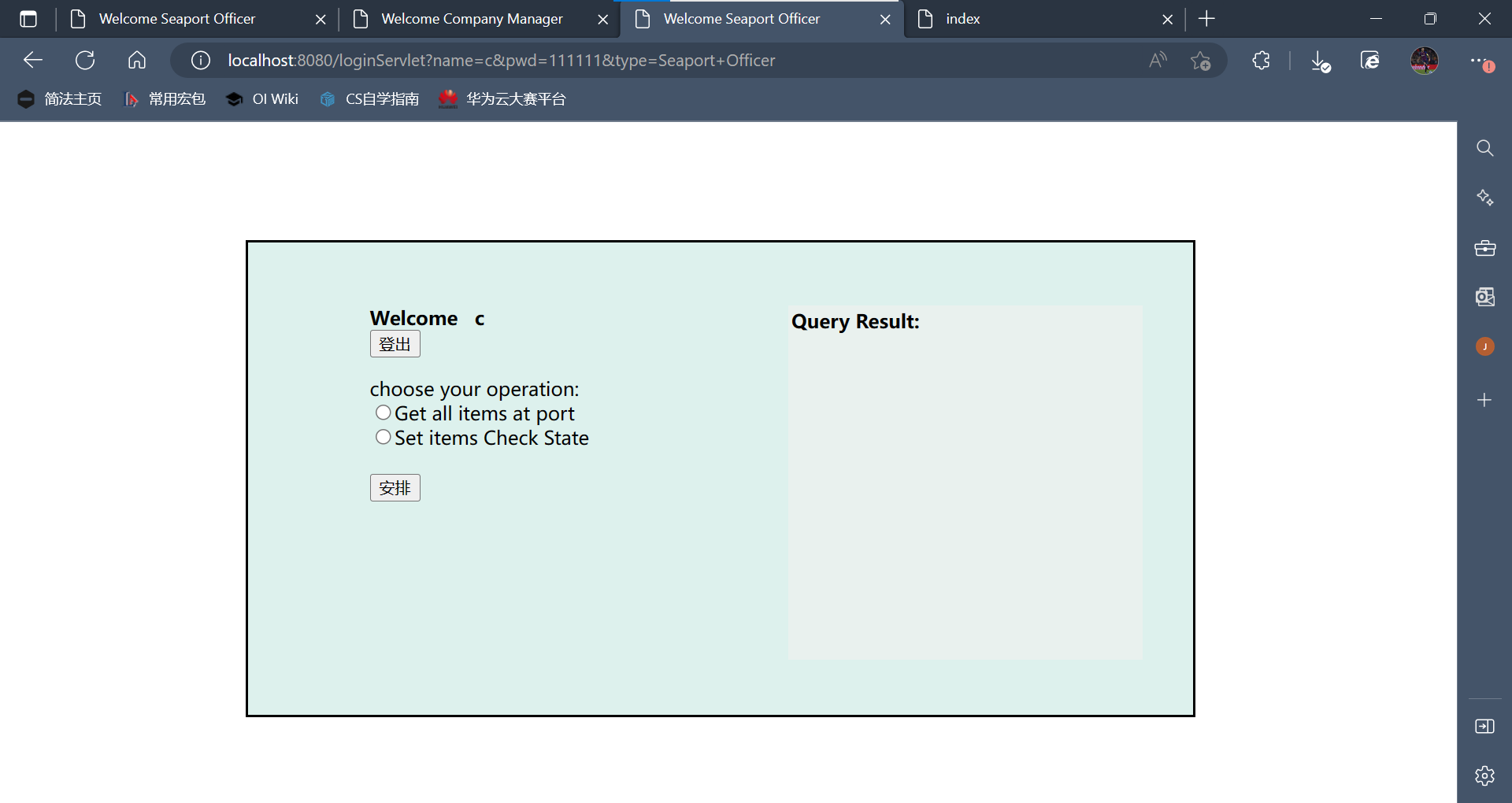


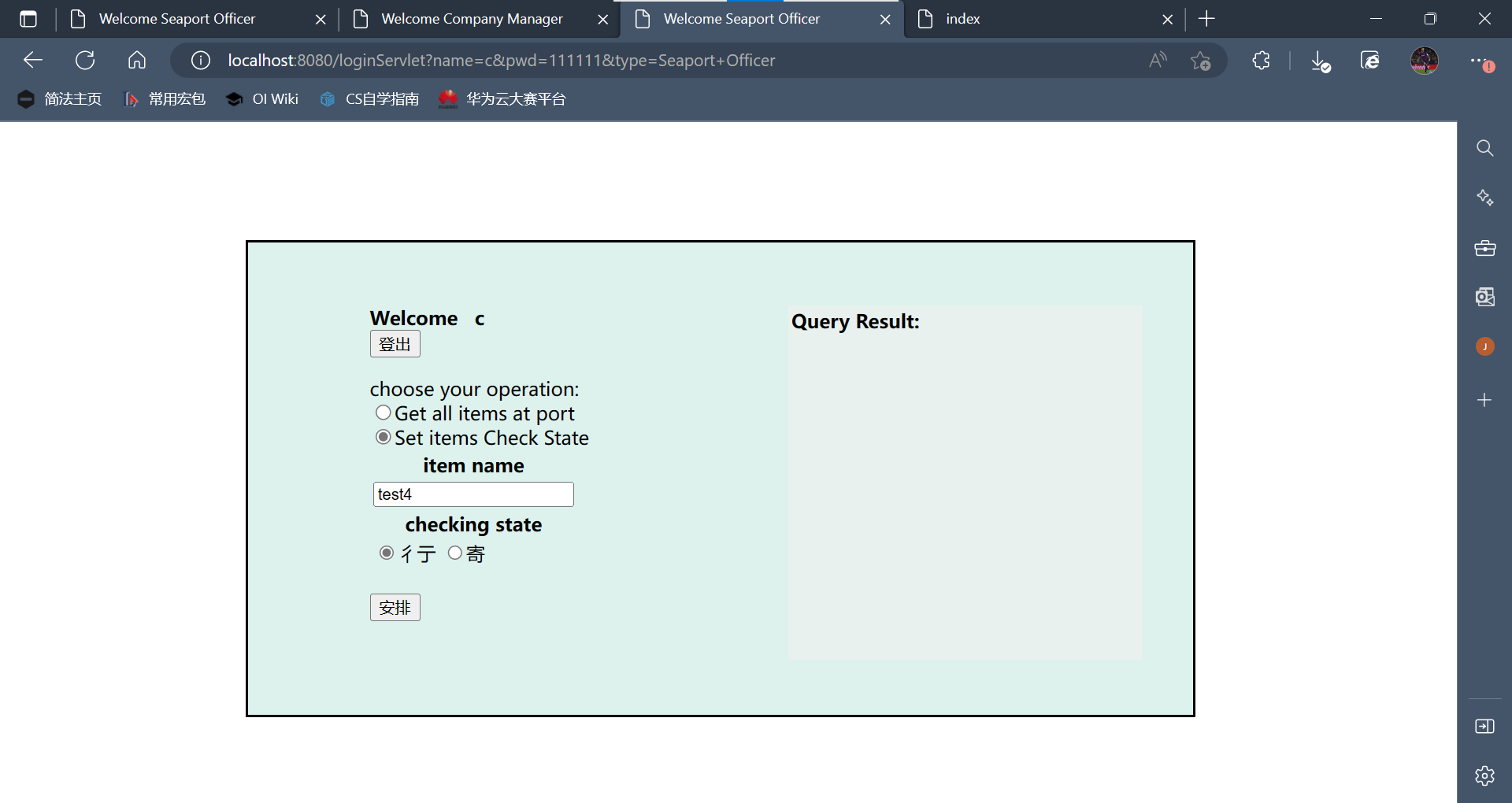
Company Manager

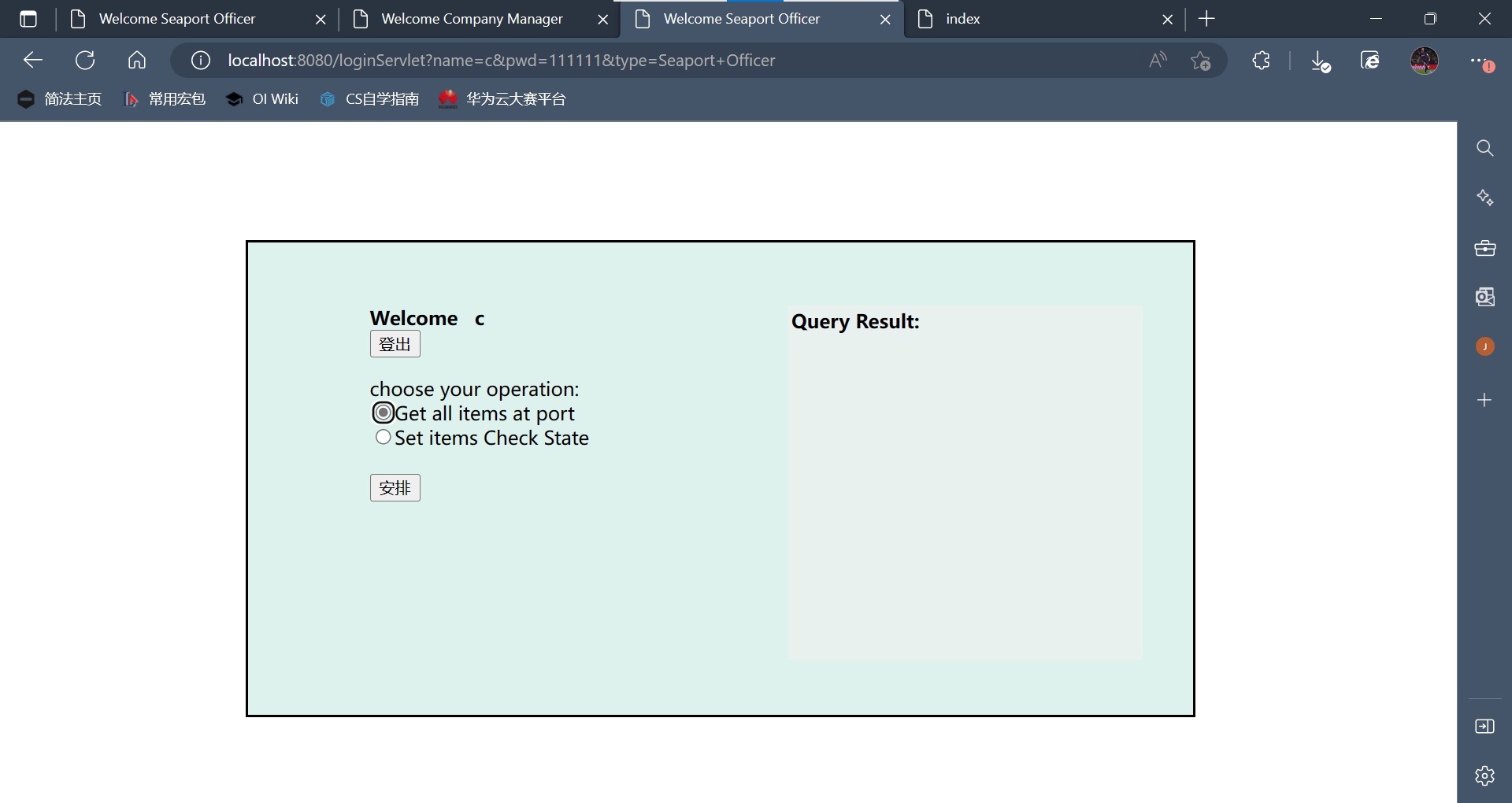


SUSTC Manager

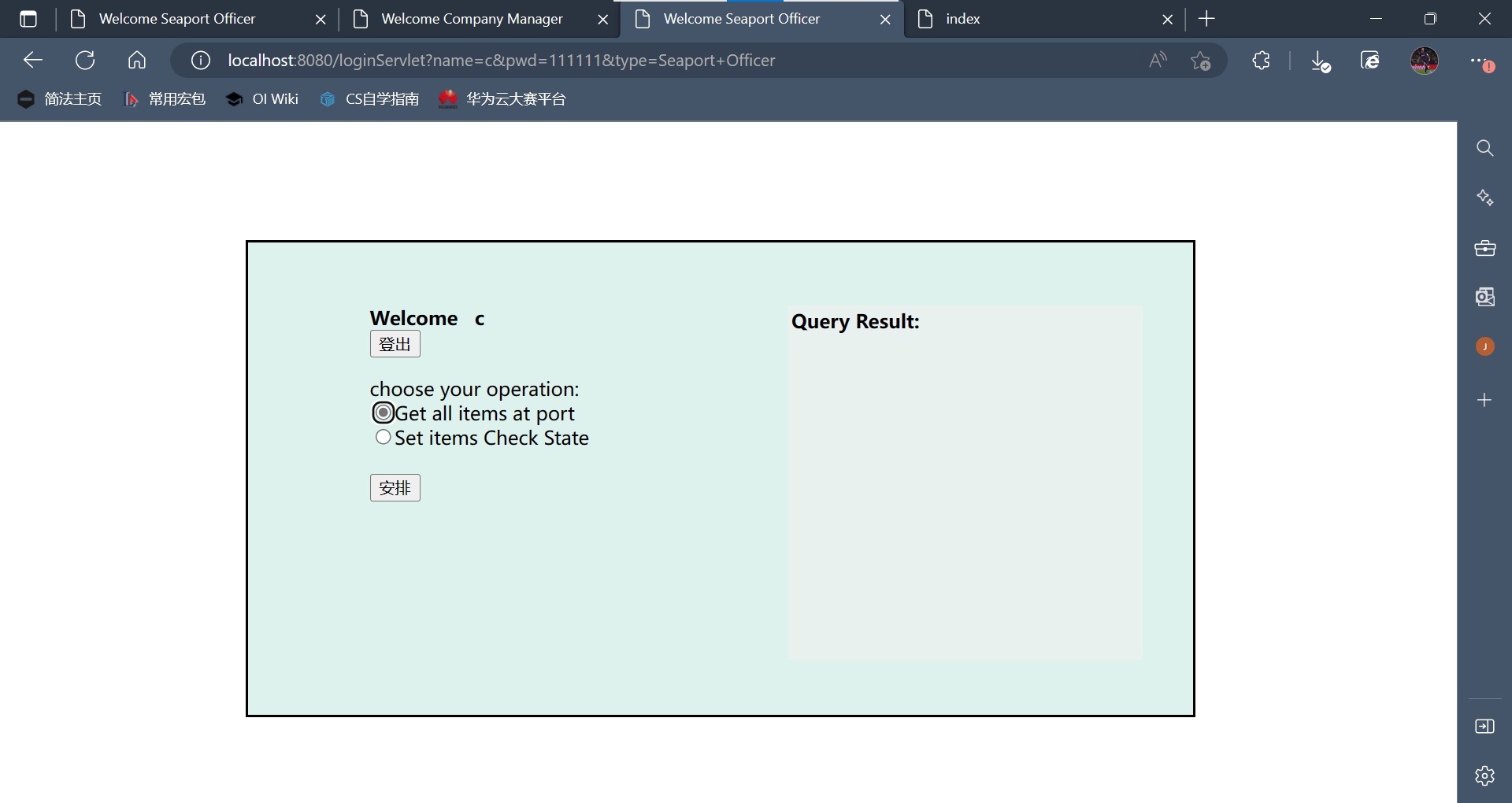
1. 下面以seaport officer为例介绍操作流程（该类型包括了一个状态更改类操作setItemCheckState和一个查询类操作getItemsAtPort，恰涵盖了本次project出现的两类操作）：
2. 初始界面，初始状态下没有进行任何选择故没有输入框也不会出现任何查询结果的信息，可以通过单击“select”组件小圆点的方式进行操作选择，在选定每个操作的同时，会浮现输入相应信息的输入框，而无关的输入框也会自动隐藏：

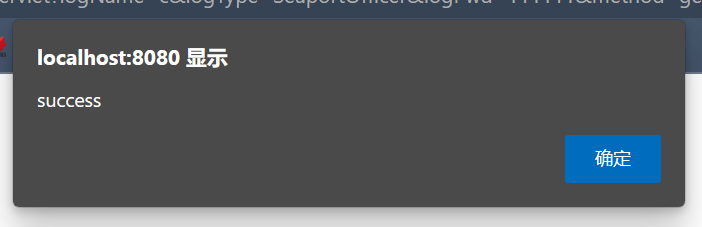




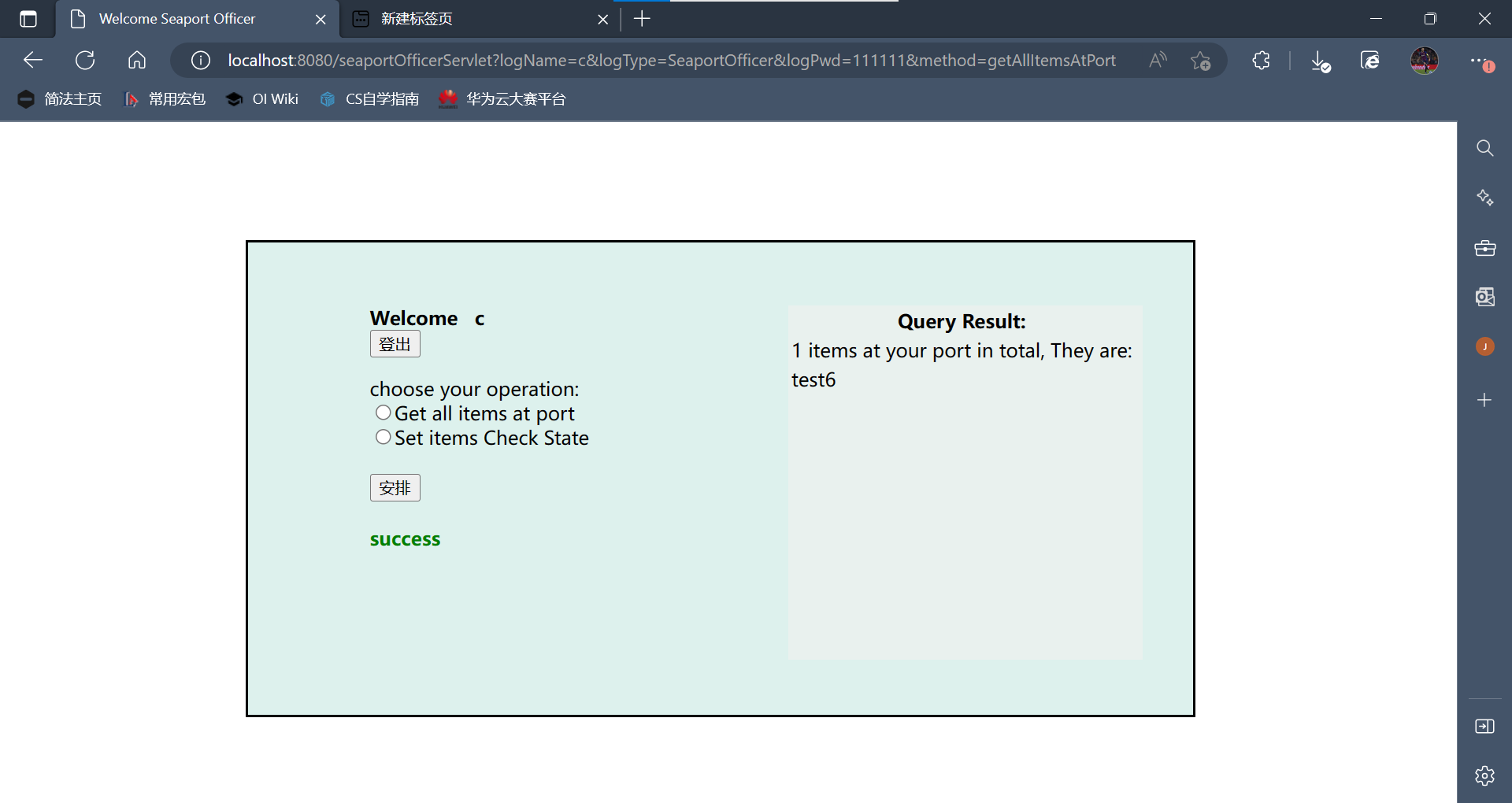


1. 查询类操作，通过输入必要参数，查询目标信息。该类操作的特点在于前端需要获取后端的返回值并展示：

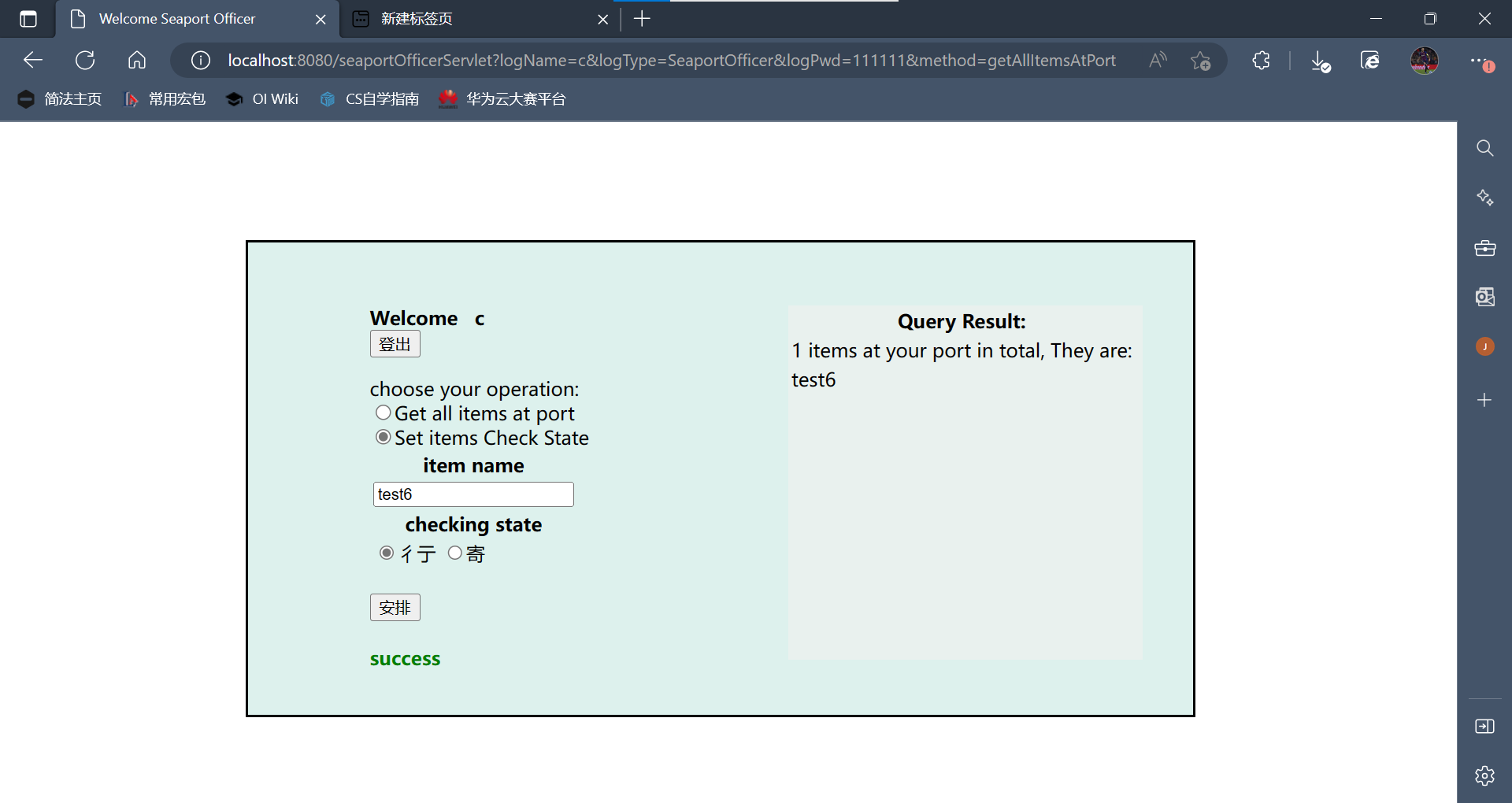




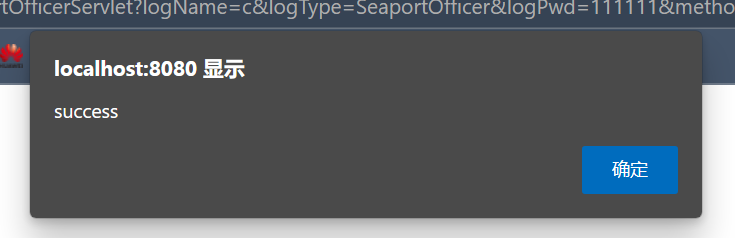
获得查询结果：



1. 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：

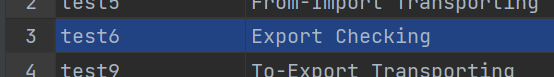


点击“安排”，弹窗显示：

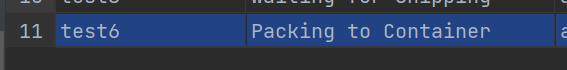


即操作成功

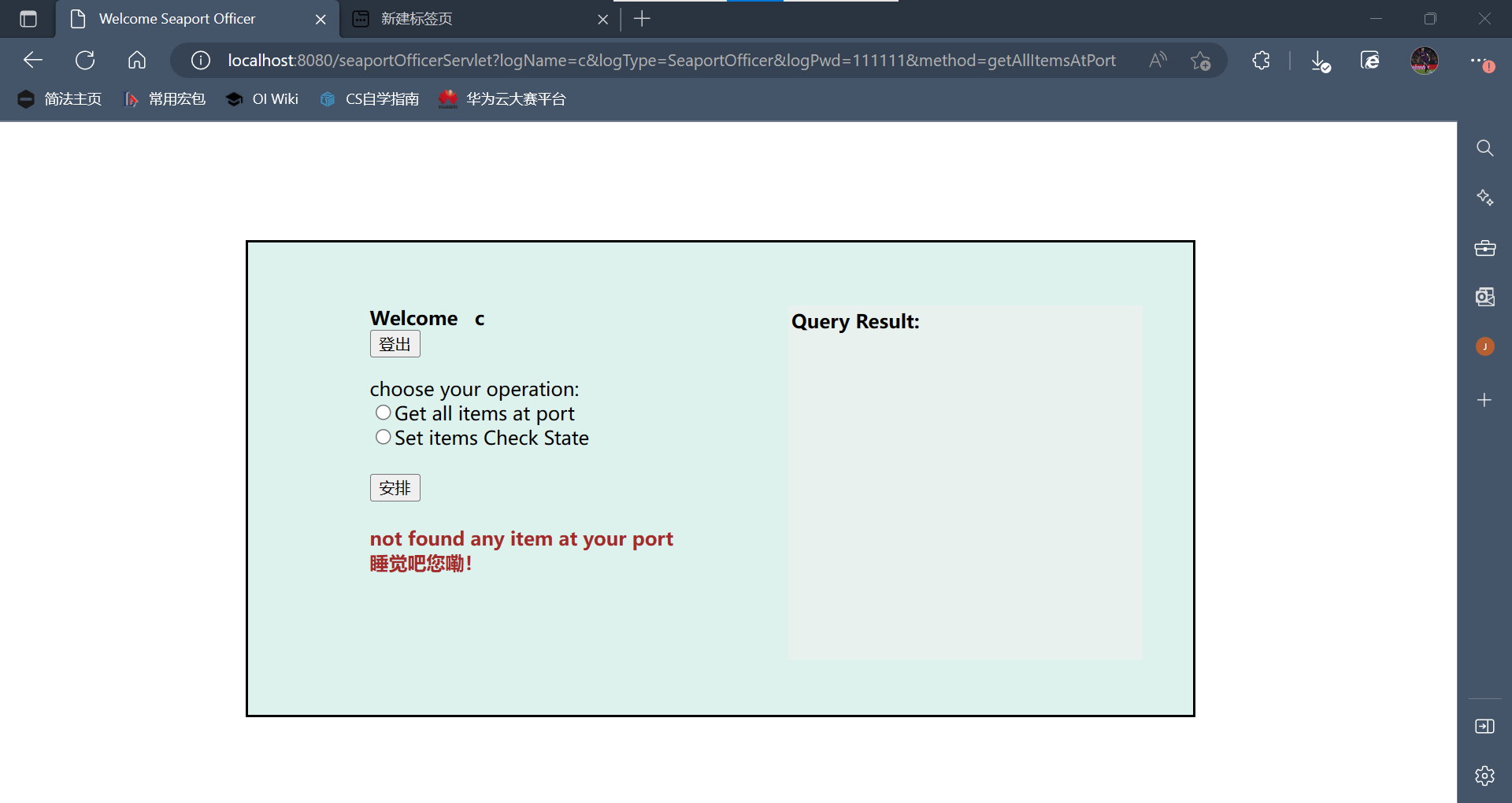
数据库中可观察到，操作前，test6状态为“Export Checking”：



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。

# 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。