Tutorial 5 J2EE

> Current Version: 3
>
> Version 1 : Designed by Yueming ZHU (in 2017)
>
> Version 2: Modified by Weiduo LIAO (in 2018)
>
> Version 3: Modified by Yueming ZHU (in 2020)

Experiment Objective

- Learn MVC framework
- Try to understand the relationship between different design layers.
- Understand the interaction between jsp and servlet.
- Learn to Use Intellij idea as Java Web Project IDE.
- Accomplish some new needs based on the given code.

Software Used

1. Install JDK

click here to download

2. Tomcat

click here to download

3. Database: PostgreSQL

click here to download

You can also use other DBMS instead.

4. Install Intellij IDEA (Ultimate)

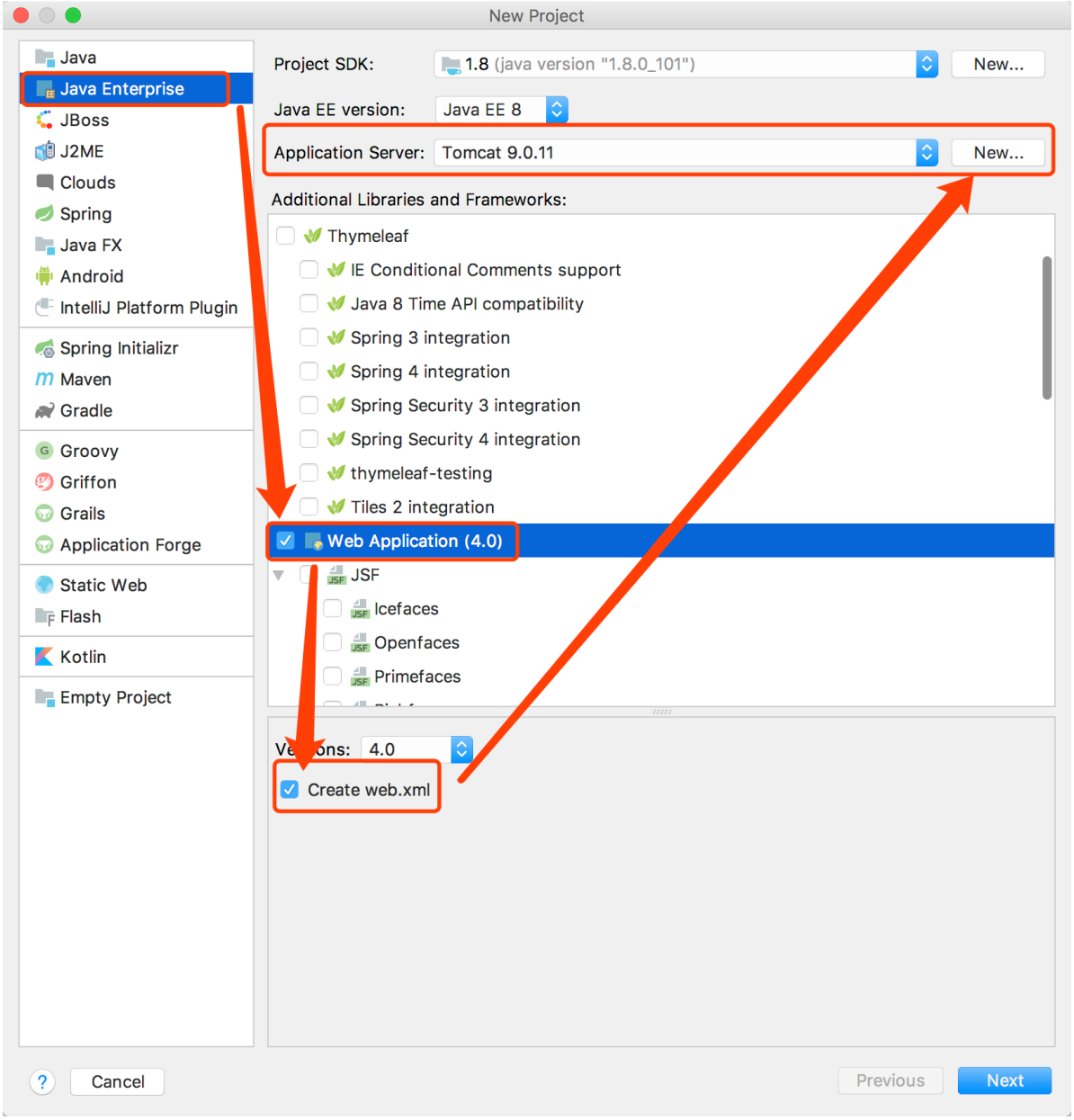
click here to download

Free student pack

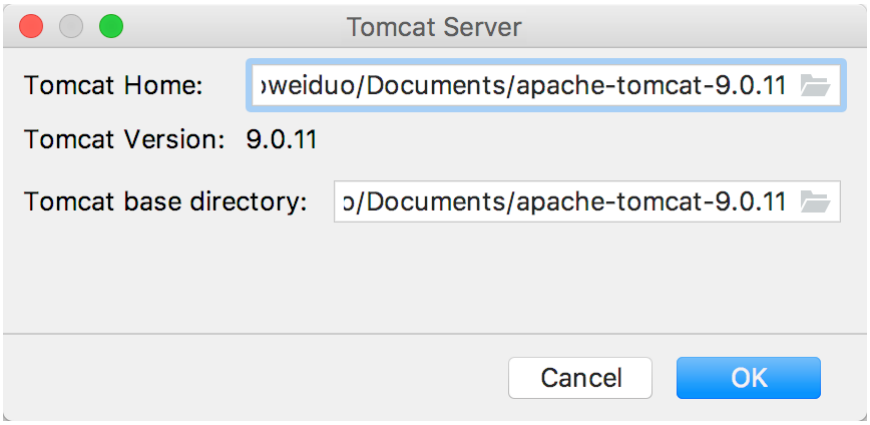
You can also use eclipse EE instead, but in this tutorial we only introduce how to build J2EE project by Intellij IDEA.

Part One: Build Project

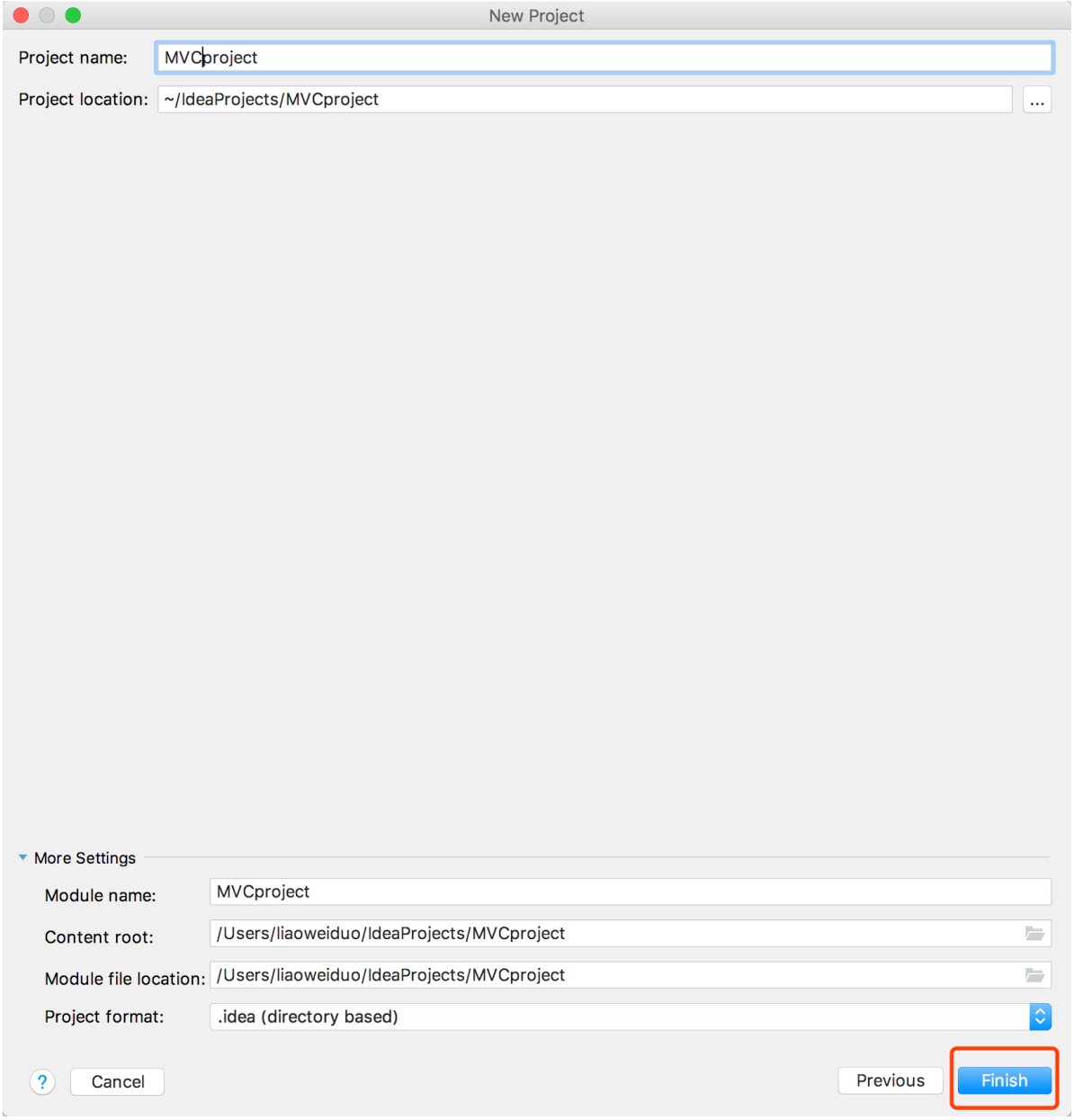
1: File -> new -> Project



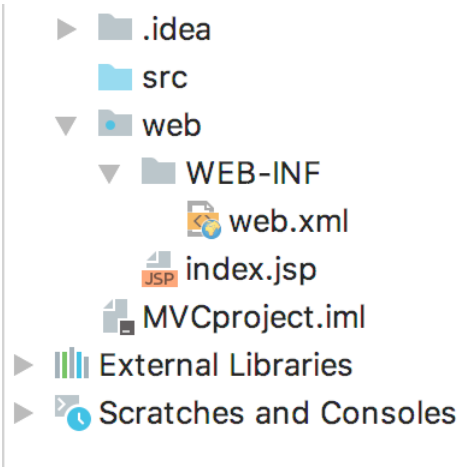
2: new -> Tomcat Server



3: next -> typing project name: MVCproject -> finish



The catalog of the project would as follows

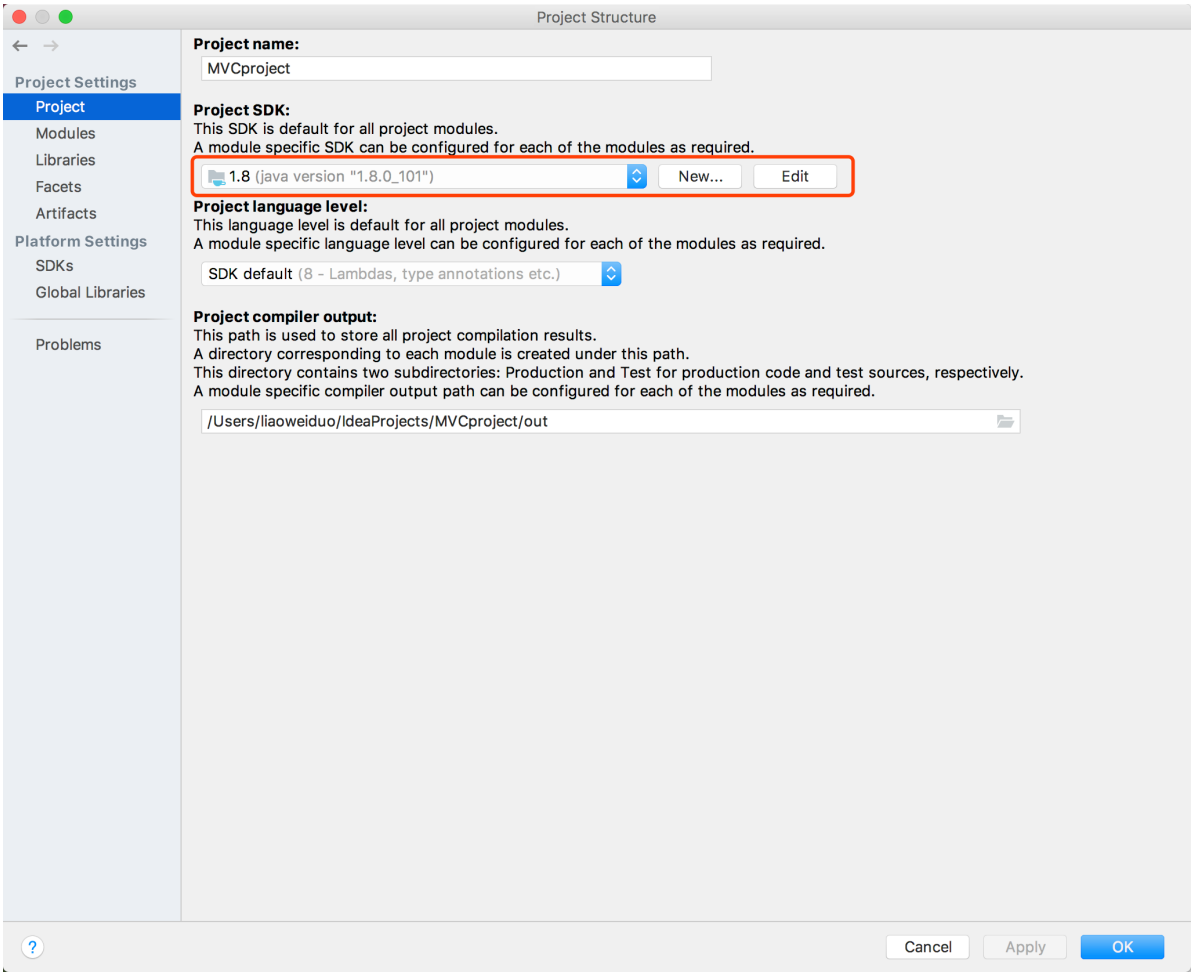


4: Create folders in WEB-INF: classes and lib.

5: In File -> Project Structure

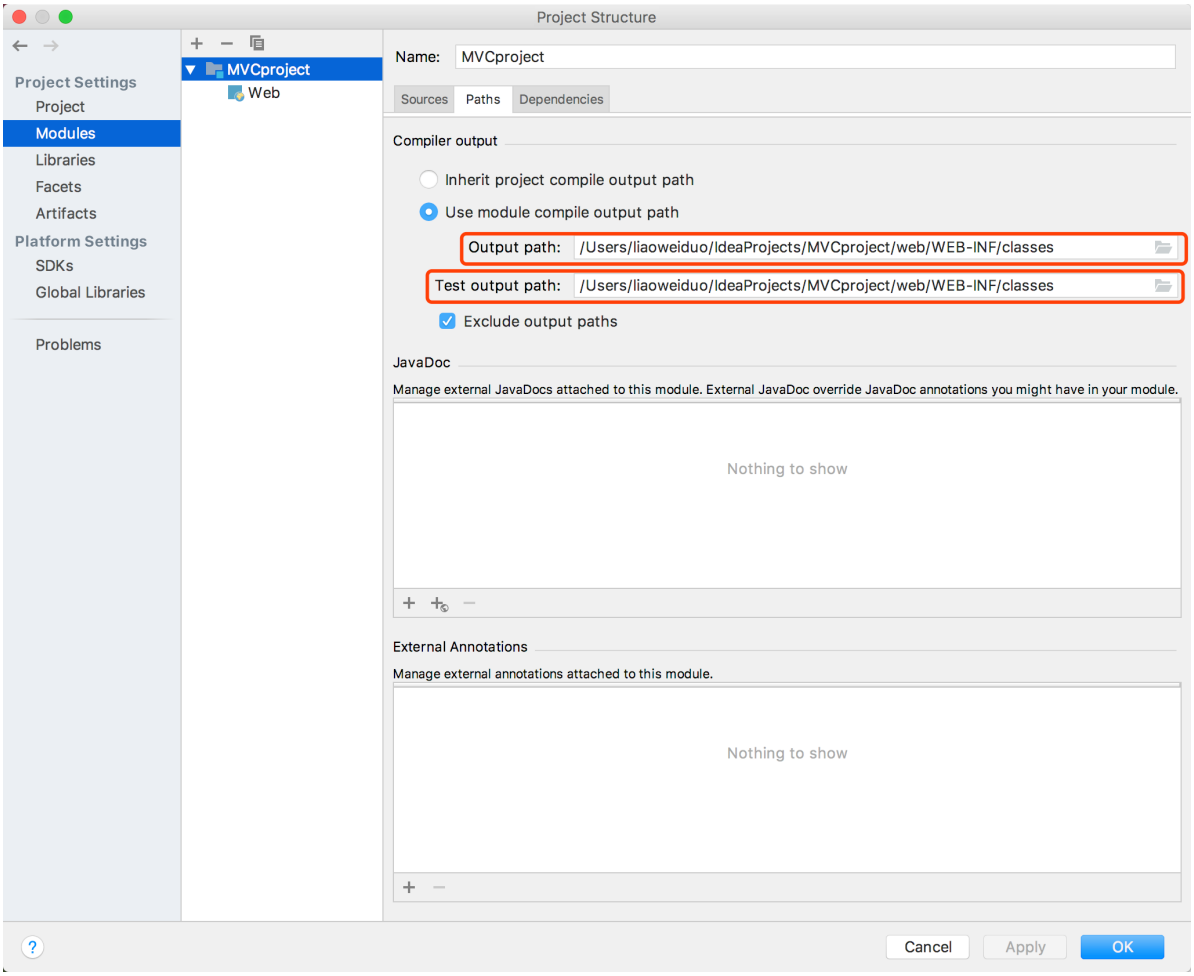
you can check your project SDK in Project tab.

(For Mac User: Right Click Project-> Open Module Settings



6: Build Module output path

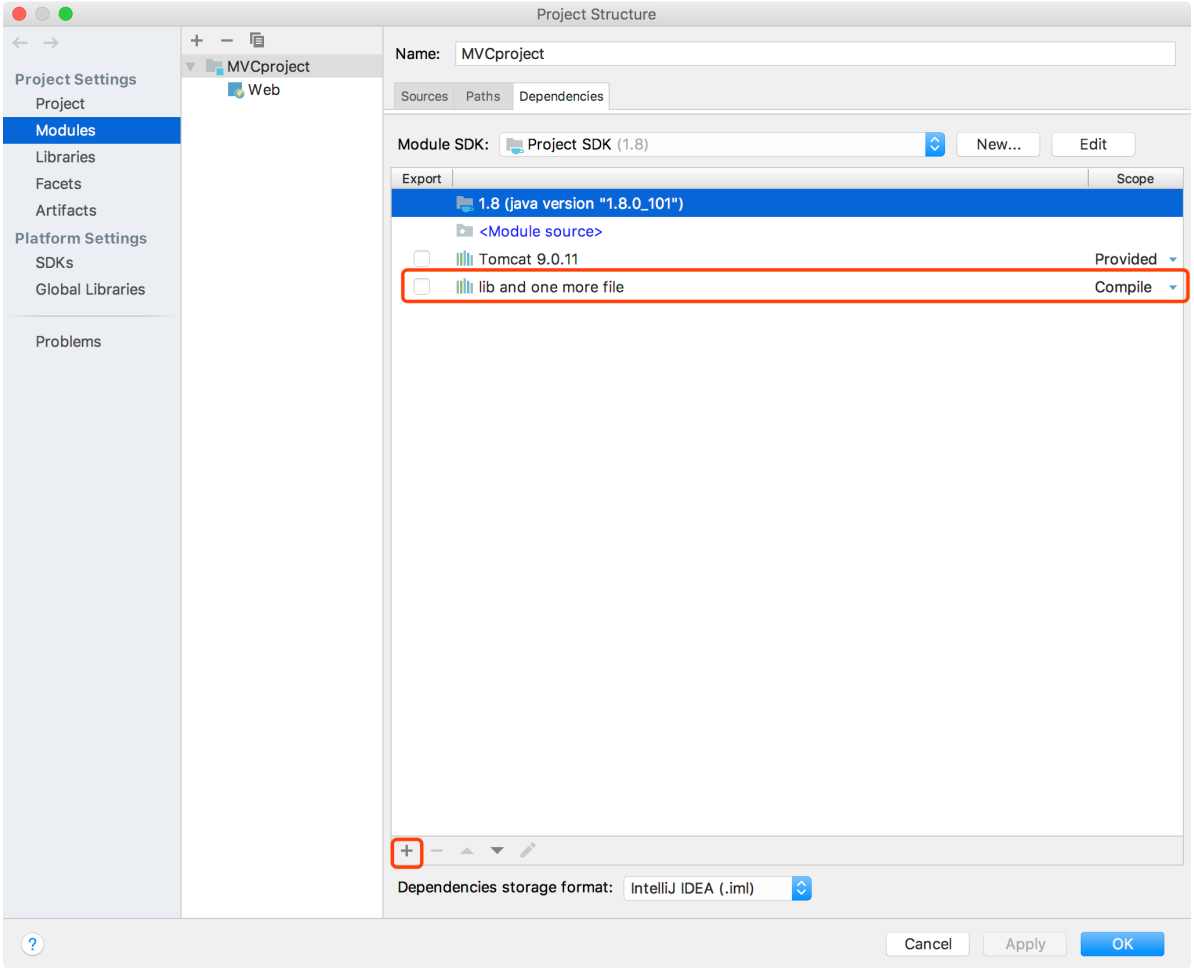
In Modules tab, change output path and test output path to classes folder you have created in step 4. The classes files built by IDE will be added into this folder.



7: Add lib folder into module dependency

In Dependencies tab, add lib folder to dependencies. You can put external dependencies like postgreSQL jdbc in this folder.

Jar or directories -> finding lib folder -> Jar Directory



8: Create First JSP

Create a `jsp` file named `login.jsp` in web folder. And edit `web.xml`: add welcome-file-list in order to modified entrance page. Write hello world in .

web.xml

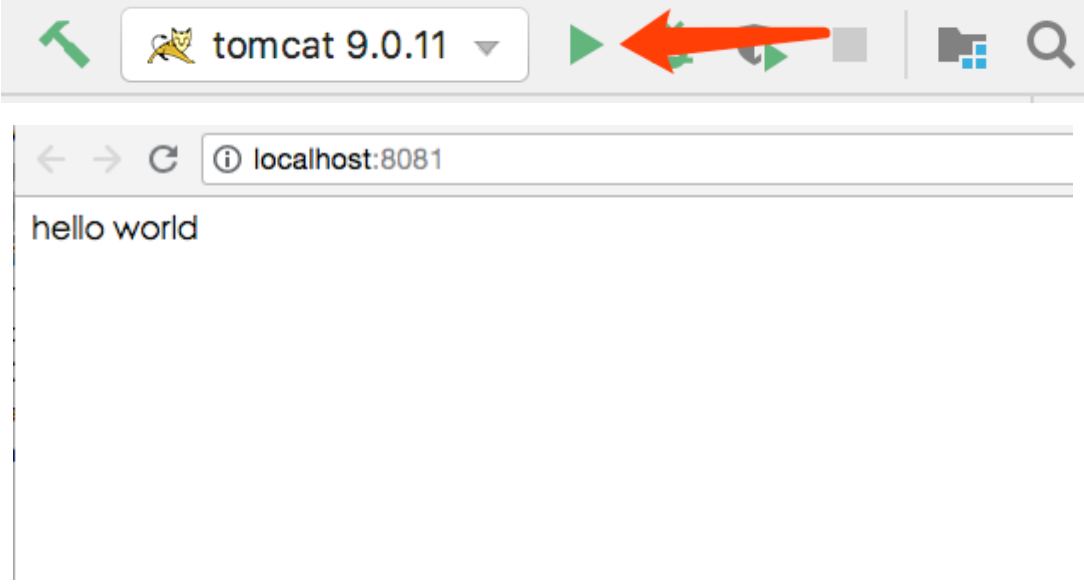
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Login.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java"
pageEncoding="UTF-8" %>
<html>
    <head>
        <meta charset="utf-8">
        <title>Test</title>
    </head>
    <body>
        Hello World!
    </body>
</html>
```

9: Start Server

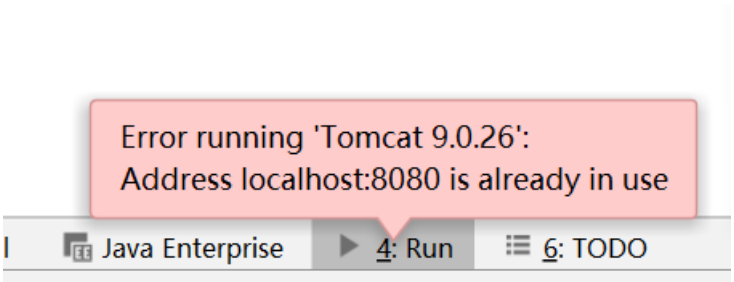
Click Run button in upper right corner. Check whether using tomcat server. Then a page showing hello world shows up.



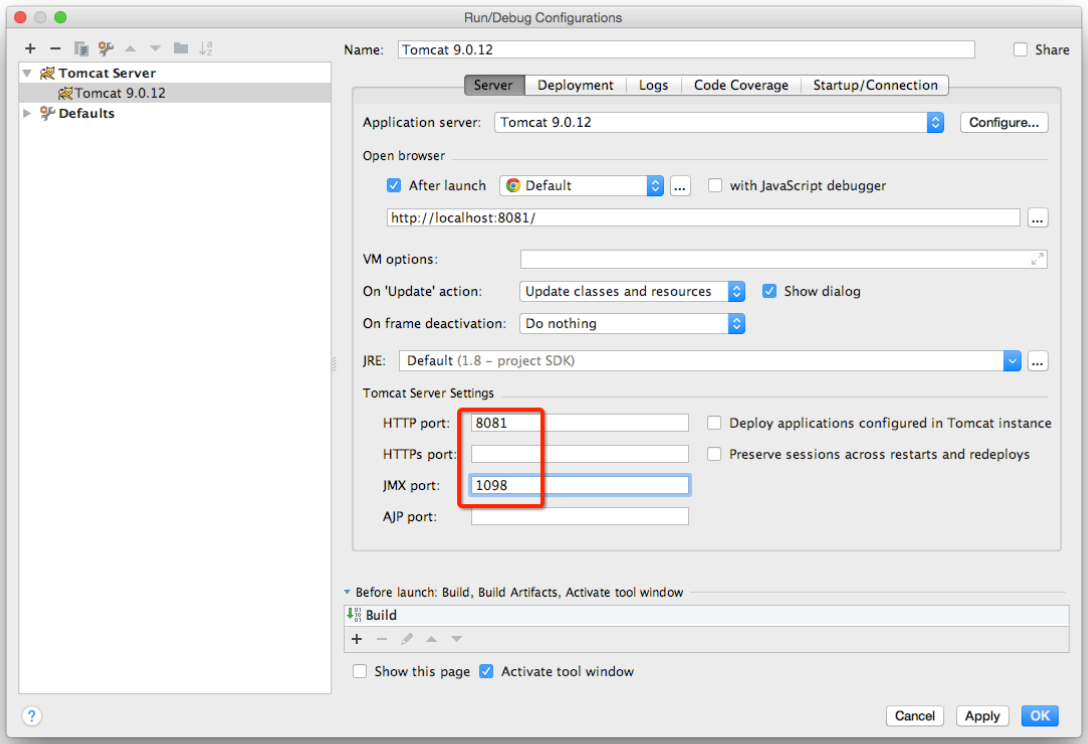
If you meet the following exception when you run the project, you need to change the **port number**.

```
08-Oct-2018 22:49:36.872 严重 [main] org.apache.catalina.util.LifecycleBase.handleSubClass
 org.apache.catalina.LifecycleException: Protocol handler initialization failed
    at org.apache.catalina.connector.Connector.initInternal(Connector.java:935)
    at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:136)
    at org.apache.catalina.core.StandardService.initInternal(StandardService.java:533)
    at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:136)
    at org.apache.catalina.core.StandardServer.initInternal(StandardServer.java:852)
    at org.apache.catalina.util.LifecycleBase.init(LifecycleBase.java:136)
    at org.apache.catalina.startup.Catalina.load(Catalina.java:633)
    at org.apache.catalina.startup.Catalina.load(Catalina.java:656) <4 internal calls>
    at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:306)
    at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:491)
Caused by: java.net.BindException: Address already in use
    at sun.nio.ch.Net.bind0(Native Method)
```

Or



Click Tomcat 9.0.12 -> Edit configurations and then change the port number.



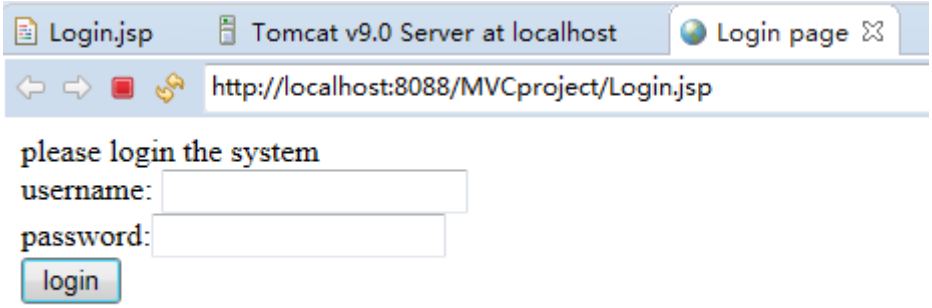
Part Two: Login

1. Create first servlet

1.1 Add login page in `login.jsp`

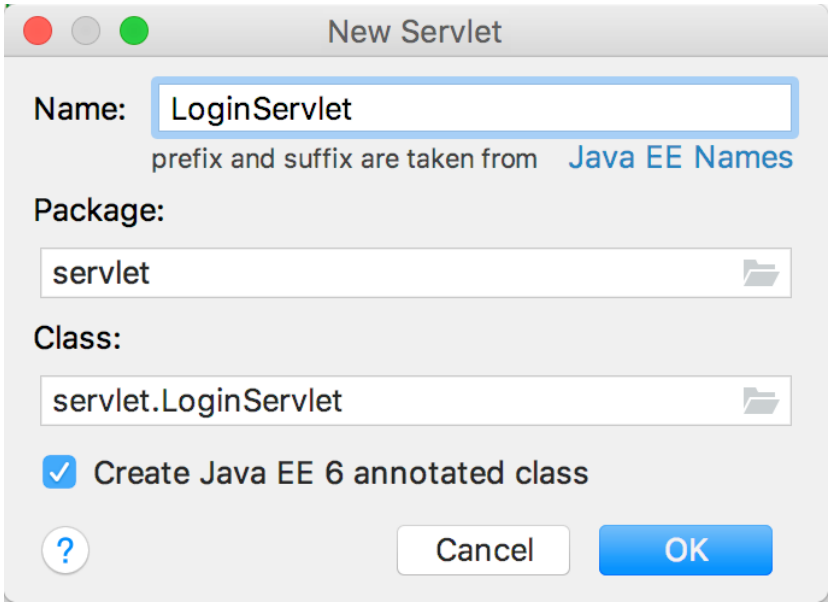
```
<body>
please login the system
<br/>
    <form action="LoginServlet" method="post">
        username: <input type="text" name="username"
                         value="<%=request.getAttribute("username") != null ?
request.getAttribute("username") : ""%>"/>
        <%=request.getAttribute("msg_username") != null ?
request.getAttribute("msg_username") : ""%>
        <br/>
        password: <input type="password" name="password"
                         value="<%=request.getAttribute("password") != null ?
request.getAttribute("password") : ""%>"/>
        <%=request.getAttribute("msg_password") != null ?
request.getAttribute("msg_password") : ""%><br/>
        <input type="submit" value="login"/>
        <br/>
        <%=request.getAttribute("msg") != null ? request.getAttribute("msg") :
""%>
    </form>
</body>
```

- After restart the server, your page would be:



1.2 Create corresponding `servlet` file of `login.jsp`.

- Firstly, create a package named servlet in src folder.
- Right click servlet package -> new -> servlet, named it LoginServlet.



- Add following code in `web.xml` to register servlet in the project so that the form in `login.jsp` can recognize the corresponding servlet

```
<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
```

The function that we want to achieve in the class `LoginServlet` is that:

(1) When username and password are all null, print both are null behind those two input boxes.

(2) If one of them is null, print null behind the null one; hold data in another.

"request" and "response" are built-in objects of `JSP`, and the server (Tomcat) can help analysis those two objects and convert them to a type of `HttpServletRequest` and `HttpServletResponse`, which can be used by servlet.

We can use `request` to get basic information of Client by using following methods, such as:

- String getMethod()

- String getRequestURL()
- String getProtocal()
- String getServletPath()
- String getServerPort()
- String getRemoteAddr()

We can also use `setArribute()` and `getArribute()` in request object. According to our tutorial, the code: `request.setAttribute(var1,var2)` in servlet can set an attribute in request, and its corresponding code: `request.getAttribute(var1)` in `jsp` can get its value `var2`

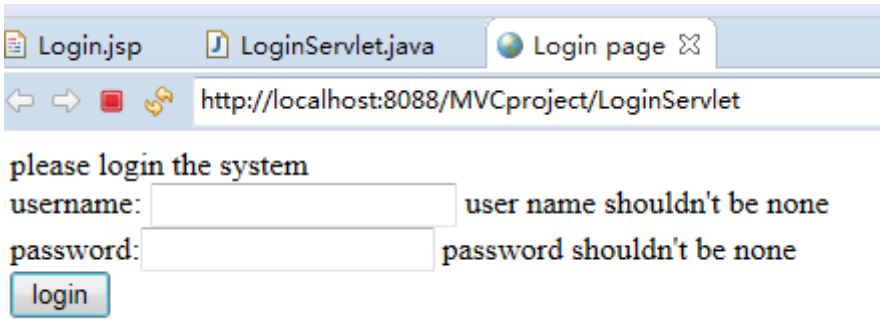
- Add code in `LoginServlet`:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    request.setCharacterEncoding("UTF-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    if ((username == null || username.equals("")) && (password == null ||
password.equals(""))) {
        request.setAttribute("msg_username", "user name shouldn't be none");
        request.setAttribute("msg_password", "password shouldn't be none");
        request.getRequestDispatcher("login.jsp").forward(request, response);
    } else if (password == null || password.equals("")) {
        request.setAttribute("username", username);
        request.setAttribute("msg_password", "password shouldn't be none");
        request.getRequestDispatcher("login.jsp").forward(request, response);
    } else if (username == null || username.equals("")) {
        request.setAttribute("password", password);
        request.setAttribute("msg_username", "user name shouldn't be none");
        request.getRequestDispatcher("login.jsp").forward(request, response);
    } else {
        System.out.println("success");
    }
}
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
```

The page is shown below:



Success if console prints success after inputting username and password.

2. Build different package layer

2.1 Create 3 packages in `src` folder

`bean`, `dao`, `service`

- **bean**

  model layer of whole project, usually being used to define entity class, data fields, manage objects and abstract objects. The data fields in it always can be mapped to the column name in table in database.

  Create a class: `UserinfoBean` in package bean, and add the **getter** and **setter** method for each private fields

  ```
  public class UserinfoBean {
      private int id;
      private String username;
      private String password;
  }
  ```

- **dao**

  data access layer, which usually design methods for database operation, and provides interface for accessing database.

  Create an interface: `UserinfoDao` in package dao:

  `public interface UserinfoDao`

  Then create a class: `UserinfoDaoImpl` implementing this interface:

  `public class UserinfoDaoImpl implements UserinfoDao`

- **service**

  Business logic layer, which doesn't interact with database directly, it is middle layer between client and database, always encapsulate business logic and provide required data to Servlet or invoke different interface that provided by `dao`.

  Create an interface: `UserinfoService` in package Service:

  `public interface UserinfoService`

  Then create a class: `UserinfoServiceImpl` implementing this interface:

  `public class UserinfoServiceImpl implements UserinfoService`

- **servlet**

  We need to connect the database to verify username and password. Now we use the four different layer to complete this login logic,

2.2 Complete the code

- In `LoginServlet.java`

  just below the statement System.**out**.println(**"success"**);, adding following code, and then import `UserinfoService` and `UserinfoServiceImpl`.

```
System.out.println("success");
UserinfoService userinfoservice = new UserinfoServiceImpl();
int result = 0;
result = userinfoservice.login(username, password);
if (result == 1) {
    System.out.println("visiting database successfully");
    //request.getRequestDispatcher("BookServlet").forward(request, response);
} else {
    request.setAttribute("msg", "the username or password is wrong");
    request.getRequestDispatcher("login.jsp").forward(request, response);
}
```

At this moment, login method will report an error, because we have not defined login method on Service layer.

- In Service layer

  Add method in `UserinfoService`:

```
int login(String username, String password);
```

  Implement login method in `UserinfoServiceImpl`, and import `UserinfoDao`, `UserinfoDaoImpl`.

```
private UserinfoDao userinfoDao = new UserinfoDaoImpl();
@Override
public int login(String username, String password) {
    int result = 0;
    try{
        result=userinfoDao.login(username,password);
    }catch(Exception e){
        e.printStackTrace();
    }
    return result;
}
```

  The login method reports an error, that's because we have not define this method in dao layer.

- In dao layer

  Add login method in `UserinforDao`

```
public int login(String username, String password) throws Exception;
```

  Implement this method in `UserinforDaoImpl`, the detailed code will be given after connect to the database.

```
@Override
public int login(String username, String password) throws Exception {
    // TODO Auto-generated method stub
          return 0;
        }
```

3 Database Connection

In this tutorial, we use `postgreSQL` as an example database to introduce, but you can use other instead.

Then we use pg command to build database:

- login to postgreSQL, in this step, please make sure your postgreSQL server is running

```
psql -d postgres -U [your DB account]
```

- Create database named `` `mvc_project `` for this tutorial

```
create database mvc_project encoding = utf8;
```

- Logout of current database, and then visit the database you created right now.

```
postgres=# \q
psql -d mvc_project -U [your DB account]
mvc_project=#
```

- create table: `userinfo` in database `mvc_project`

```
create table userinfo(
    id serial not null primary key ,
    username varchar(20) not null,
    password varchar(20) not null
);
```

- import several data into `userinfo`

```
mvc_project=# insert into userinfo(username, password) values('成龙',123);
INSERT 0 1
mvc_project=# insert into userinfo(username, password) values('范冰冰',123);
INSERT 0 1
mvc_project=# insert into userinfo(username, password) values('zym',123);
INSERT 0 1
mvc_project=#  select * from userinfo;
 id | username | password
----+----------+----------
  1 | 成龙     | 123
  2 | 范冰冰   | 123
  3 | zym    | 123
(3 行记录)
```

4 Complete the code

- Create package: `util` in `src` folder. Create `DBUtil.java` in package `util`. this class defines methods of connecting to the database and closing the database.

```
public class DBUtil {
   private Connection connection;
    private String host = "127.0.0.1";
    private String dbname = "mvc_project";
    private String port = "5432";
    private String name = "?";
    private String password = "?";

    public Connection getConnection() throws Exception {

        try {
            Class.forName("org.postgresql.Driver");

        } catch (Exception e) {
            System.err.println("Cannot find the PostgreSQL driver.");
            return null;
        }

        try {
            String url = "jdbc:postgresql://" + host + ":" + port + "/" +
dbname;
            this.connection = DriverManager.getConnection(url, name, password);
        } catch (SQLException e) {
            System.err.println("Database connection failed");
            System.err.println(e.getMessage());
        }
        return connection;
    }

    public void closeDBResource(Connection connection,
                                PreparedStatement preparedStatement,
                                ResultSet resultSet) {
        try {
            if (resultSet != null) {
                resultSet.close();
            }
            if (preparedStatement != null) {
                preparedStatement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

- After that, implement method `login` in `Dao` layer.

```
    Connection connection=null;
    DBUtil dbutil = new DBUtil();
```

```
    ResultSet resultSet =null;
    PreparedStatement preparedStatement = null;

    @Override
    public int login(String username, String password) throws Exception {
        int result = 0;
        connection = dbutil.getConnection();
        String sql = "select count(*) from userinfo where username=? and
password=?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, username);
        preparedStatement.setString(2, password);
        resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            result = resultSet.getInt(1);
        }
        dbutil.closeDBResource(connection, preparedStatement, resultSet);
        return result;

    }
```

- After that, implement method `login` in `service` layer.

```
 @Override
    public int login(String username, String password) {
        int result = 0;
        try{
            result=userinfoDao.login(username,password);
        }catch(Exception e){
            e.printStackTrace();
        }
        return result;
    }
```

- save files and click run tomcat, the result shows below:

```
visiting database successfully
```

Part Three: Register

1. Create JSP page

Create `jsp` page named `register.jsp`, and add following codes in .

```
Please register a user
    <br>
    <form action="RegisterServlet" method="post">
        username: <input type="text" name="username"
                         value="<%=request.getAttribute("username") != null ?
request.getAttribute("username") : ""%>" />
        <%=request.getAttribute("msg_username") != null ?
request.getAttribute("msg_username") : ""%>
        <br /> password:<input type="password" name="password"
                               value="<%=request.getAttribute("password") !=
null ? request.getAttribute("password") : ""%>" />
```

```
        <%=request.getAttribute("msg_password") != null ?
request.getAttribute("msg_password") : ""%><br />
        confirm password:<input type="password" name="con_password"
                                value="<%=request.getAttribute("con_password")
!= null ? request.getAttribute("con_password") : ""%>" />
        <%=request.getAttribute("msg_con_password") != null ?
request.getAttribute("msg_con_password") : ""%><br />
        <input type="submit" value="Register" />
    </form>
    <a href="login.jsp">Return back to login page</a>
    <%=request.getAttribute("msg") != null ? request.getAttribute("msg") : ""%>
```

There are three input tags, which are handled by a form, including `username`, `password` and `confirm password` in this page, which will be sent to `servlet` (marked by action) by **post** method after click submit button.

2. Create servlet

Create servlet `RegisterServlet.java` in package `servlet`. (Do not forget registering this `servlet` in `web.xml`, method given here)

What we needed to accomplish is as following:

- If username and password are both null, print information after two input box.
- If username is null only, print that username should not be null and clear password.
- If password is null only, print that password should not be null but hold username.
- If username and password are both filled, but confirm password is null, print that confirm password should not be null and hold username and password.
- If three inputs are all filled, but with different values between password and confirm password, print that two passwords are different and clear confirm password

```
 protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String conPassword = request.getParameter("con_password");
        if (username == null) {
            username = "";
        }
        if (password == null) {
            password = "";
        }
        if (conPassword == null) {
            conPassword = "";
        }

        if (username.equals("") && password.equals("")) {
            request.setAttribute("msg_username", "user name shouldn't be none");
            request.setAttribute("msg_password", "password shouldn't be none");
            request.getRequestDispatcher("register.jsp").forward(request,
response);
```

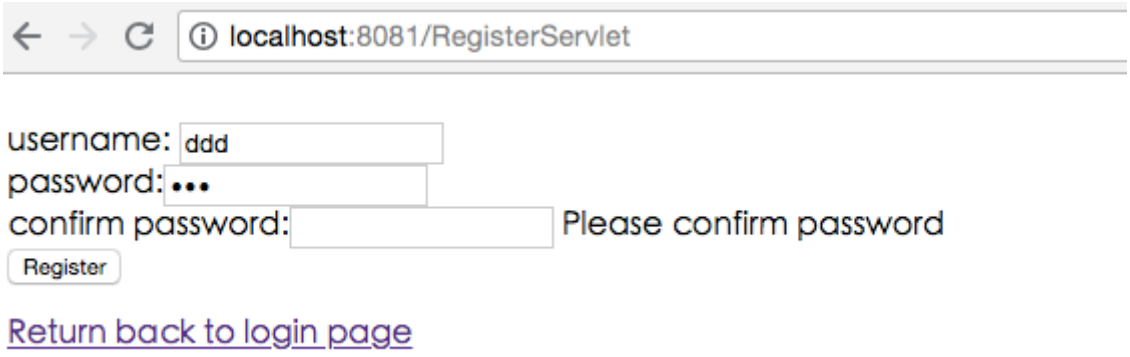
```
        } else if (username.equals("")) {
            request.setAttribute("password", password);
            request.setAttribute("msg_username", "user name shouldn't be none");
            request.getRequestDispatcher("register.jsp").forward(request,
response);
        } else if (password.equals("")) {
            request.setAttribute("username", username);
            request.setAttribute("msg_password", "password shouldn't be none");
            request.getRequestDispatcher("register.jsp").forward(request,
response);
        } else {
            if (conPassword.equals("")) {
                request.setAttribute("username", username);
                request.setAttribute("password", password);
                request.setAttribute("msg_con_password", "Please confirm
password");
                request.getRequestDispatcher("register.jsp").forward(request,
response);
            } else {
                if (!conPassword.equals(password)) {
                    request.setAttribute("password", password);
                    request.setAttribute("username", username);
                    request.setAttribute("msg_con_password", "Two password is
not same");
                    request.setAttribute("con_password", "");

 request.getRequestDispatcher("register.jsp").forward(request, response);
                } else {
                    System.out.println("Success");
                }
            }

        }

    }
```

Then start the server, and input the url: `localhost:8081/RegisterServlet`, we will get a "**Success**" in console window if we enter according to the above specifications



3. Complete code in three layers:

3.1 Modify servlet

We need encapsulate username and password into userinfo object. Create a `userinfoBean` object, and add information to `userinfoBean`, passing it to inner layers.

Add following code after `System.out.println("Success");`

```
UserinfoService userinfoService=new UserinfoServiceImpl();
UserinfoBean userinfoBean=new UserinfoBean();
userinfoBean.setUsername(username);
userinfoBean.setPassword(password);
int result=userinfoService.registerUserinfo(userinfoBean);
System.out.println("In servlet"+result);
if(result==1){
    request.setAttribute("msg", "Register is success, please login to the
system");
    request.setAttribute("username", username);
    request.getRequestDispatcher("login.jsp").forward(request, response);
}else{
    request.setAttribute("msg", "Register is failed");
    request.getRequestDispatcher("register.jsp").forward(request, response);
}
```

3.2 Modify Service layer

It will report an error on method registerUserinfo, for the reason that we haven't defined this method in Service layer.

Create method `registerUserinfo` in service interface and corresponded implementation in `userinfoServiceImpl`

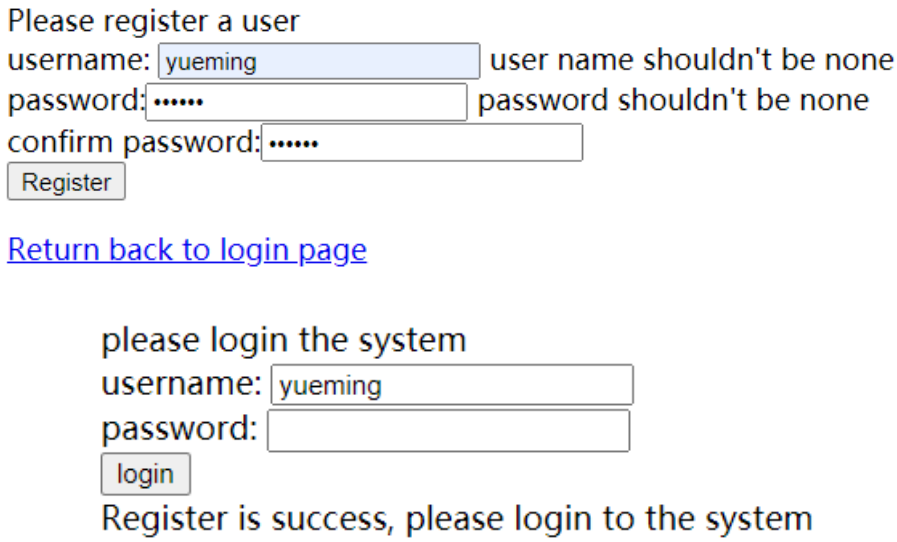
```
@Override
public int registerUserinfo(UserinfoBean userinfoBean) {
    int result=0;
    try{
        result=userinfoDao.registerUserinfo(userinfoBean);
    }catch(Exception e){
        e.printStackTrace();
    }
    return result;
}
```

3.3 Modify Dao layer

The same error of the method `registerUserinfo`. We need to create this method on `dao` layer, and then add following code.

```
@Override
    public int registerUserinfo(UserinfoBean userinfoBean) throws Exception {
        int result = 0;
        connection = dbutil.getConnection();
        String sql = "insert into userinfo (username, password) values (?,?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, userinfoBean.getUsername());
        preparedStatement.setString(2, userinfoBean.getPassword());
        result = preparedStatement.executeUpdate();
        dbutil.closeDBResource(connection, preparedStatement, resultSet);
        return result;
    }
```

Then you can restart your server, and the page would be as follows:



Part Four : List all Book

After login in, it will jump to another page about a list of books' information. We will implement this function step by step.

1. Create Book table in database

```
create table book
(
    id        serial      not null
            primary key,
    book_name varchar(30) not null,
    author varchar(20) not null,
    price double precision not null ,
    date_added varchar(10)
);
```

Then import several data into book

```
insert into book (book_name,author,price,date_added) values
('C++','Emy',79.5,'2020-11-1');
insert into book (book_name,author,price,date_added) values
('J2EE','Mary',150,'2020-09-10');
insert into book (book_name,author,price,date_added) values
('SQL','He',500,'2020-08-09');
insert into book (book_name,author,price,date_added) values
('Java','Lili',99,'2020-11-09');
```

After execute select all, it will be:

```
mvc_project=# select * from book;
 id | book_name | author | price | date_added
----+-----------+--------+-------+------------
  1 | C++       | Emy    |  79.5 | 2020-11-1
  2 | J2EE      | Mary   |   150 | 2020-09-10
  3 | SQL       | He     |   500 | 2020-08-09
  4 | Java      | Lili   |    99 | 2020-11-09
(4 行记录)
```

2. Build different layer of book

According to the former work about building framework about `userinfo`, we need to do same similar operation of book.

Firstly we need create several .java files:

(1) Bean layer: BookBean and its attribute with getter and setter:

```
private int id;
private String bookName;
private String author;
private double price;
private String addingDate;
```

(2) Dao layer: BookDao and BookDaoImpl (implements BookDao)

(3) Service layer: BookService and BookServiceImpl (implements BookService)

(4) Servlet layer: BookServlet (do not forget registering in web.xml)

3. Build Front End (JSP)

Create `BookList.jsp`, and then add following code in :

```
<%
    List<BookBean> bookBeanList = (List<BookBean>)
request.getAttribute("bookList");
%>
    <table>
        <caption id="a"><h2>BookList</h2></caption>
        <tr>
            <th>BookID</th>
            <th>BookName</th>
            <th>Author</th>
            <th>Price</th>
```

```
            <th>Adding Date</th>
            <th>Operation</th>
        </tr>

        <%
            if (bookBeanList != null && bookBeanList.size() > 0) {
                for (int i = 0; i < bookBeanList.size(); i++) {
        %>
        <tr>
            <td><%=i + 1%>
            </td>
            <td><%=bookBeanList.get(i).getBookName()%>
            </td>
            <td><%=bookBeanList.get(i).getAuthor()%>
            </td>
            <td><%=bookBeanList.get(i).getPrice() %>
            </td>
            <td><%=bookBeanList.get(i).getAddingDate() %>
            </td>
            <td><a href="#">delete</a></td>
        </tr>
        <%
            }
        } else {
        %>
        <tr>
            <td colspan="6">can not get the book infomation</td>
        </tr>
        <%
            }
        %>
    </table>
```

`<a href="#">` is a link that would be added in our following work about delete function.

At the top of `jsp` file, add following two lines to import those two classes.

```
<%@ page import="bean.BookBean" %>
<%@ page import="java.util.List" %>
```

4. Complete Servlet

After copy following code, do not forget to import related classes into the servlet.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        BookService bookService=new BookServiceImpl();
        List<BookBean> bookList=bookService.fetchBookList();
        request.setAttribute("bookList", bookList);
        request.getRequestDispatcher("BookList.jsp").forward(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request,response);
    }
```

5. Complete code in other layers

5.1 Service

Add following code in `BookServiceImpl.java`

```
 private BookDao bookDao=new BookDaoImpl();
    @Override
    public List<BookBean> fetchBookList() {
        List<BookBean> bookList=null;
        try{
            bookList=bookDao.fetchBookList();
        }catch(Exception e){
            e.printStackTrace();
        }
        return bookList;
    }
```

5.2 Dao

Add following code in `BookDaoImpl.java`

```
 private DBUtil dbutil = new DBUtil();
    private Connection connection = null;
    private PreparedStatement preparedStatement = null;
    private ResultSet resultSet = null;

    @Override
    public List<BookBean> fetchBookList() throws Exception {
        List<BookBean> bookBeanList= new ArrayList<>();
        connection = dbutil.getConnection();
        String sql = "select * from book";
        preparedStatement = connection.prepareStatement(sql);
        resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            BookBean bookBean = new BookBean();
            bookBean.setId(resultSet.getInt("id"));
            bookBean.setBookName(resultSet.getString("book_name"));
            bookBean.setAuthor(resultSet.getString("author"));
            bookBean.setPrice(resultSet.getDouble("price"));
            bookBean.setAddingDate(resultSet.getString("date_added"));
            bookBeanList.add(bookBean);
        }
```

```
        dbutil.closeDBResource(connection, preparedStatement, resultSet);
        return bookBeanList;
    }
```

5.3 Modify `LoginServlet`

Implement page dispatch function: Add the statement below just under the `"visiting database successfully"`.

```
request.getRequestDispatcher("BookServlet").forward(request, response);
```

Restart project, when login successfully, the page would be dispatch from `BookServlet` and then shown as follows:

BookList

BookID	BookName	Author	Price	Adding Date	Operation
1	C++	Emy	79.5	2020-11-1	delete
2	J2EE	Mary	150.0	2020-09-10	delete
3	SQL	He	500.0	2020-08-09	delete
4	Java	Lili	99.0	2020-11-09	delete

Part Five: Delete Book

1. Modified `BookList.jsp`

(1) `<a href="#">` add a `DeleteBookServlet` to implement delete function.

```
<td><a href="DeleteBookServlet?id=<%=bookBeanList.get(i).getId()%>">delete</a>
</td>
```

(2) add an output after

, in order to show the result information of operation.

```
<%=request.getAttribute("msg") != null ? request.getAttribute("msg") : ""%>
```

we pass the id as parameter in `DeleteBookServlet.java`, which is the primary key in database.

2. Servlet layer

Create `DeleteBookServlet.java` (do not forget registering in `web.xml`), adding following code:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        try{
            String idString=request.getParameter("id");
```

```
            int id=Integer.parseInt(idString);
            BookService bookService=new BookServiceImpl();
            int result=bookService.deleteBookById(id);
            if(result==1){
                request.setAttribute("msg", "delete successfully");
            }else{
                request.setAttribute("msg", "delete failed");
            }
            request.getRequestDispatcher("BookServlet").forward(request,
response);
        }catch(Exception e){
            request.setAttribute("msg", "The id of this user is null");
            request.getRequestDispatcher("BookServlet").forward(request,
response);
        }
    }
```

3. Service

Add following code in `BookServiceImpl.java`

```
@Override
    public int deleteBookById(int id) {
        int count = 0;
        try {
            count = bookDao.deleteBookById(id);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return count;
    }
```

4. Dao

Add following code in `BookDaoImpl.java`

```
@Override
    public int deleteBookById(int id) throws Exception {
        int result;
        connection = dbutil.getConnection();
        String sql = "delete from book where id=?";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt(1, id);
        result = preparedStatement.executeUpdate();
        dbutil.closeDBResource(connection, preparedStatement, resultSet);
        return result;
    }
```

Restart project, after we delete all books successfully, the page would be shown as follows

BookList

BookID BookName Author Price Adding Date Operation

can not get the book infomation