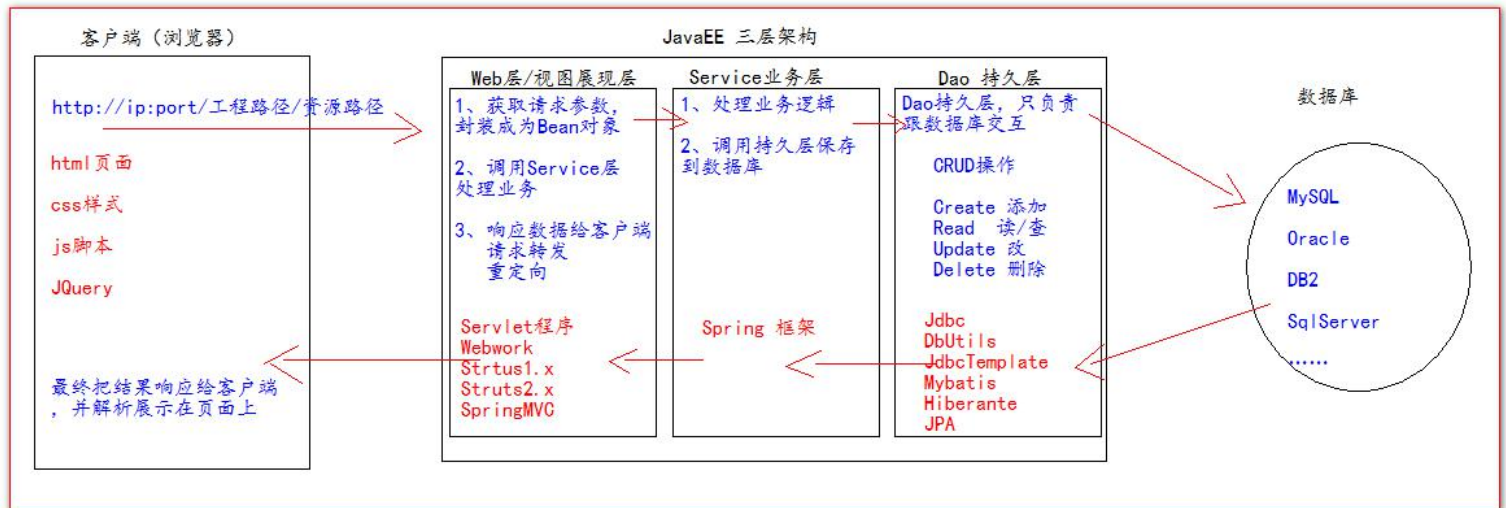


书城第二阶段——用户注册和登陆

讲师：王振国

今日任务

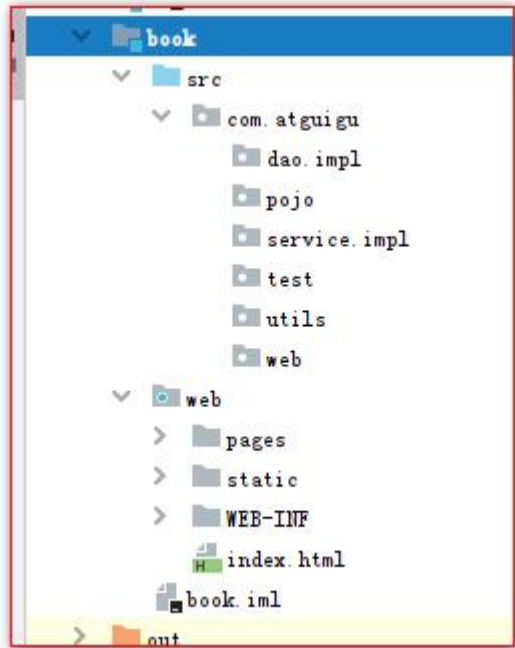
1.JavaEE 项目的三层架构



分层的目的是为了解耦。解耦就是为了降低代码的耦合度。方便项目后期的维护和升级。

web 层	com.atguigu.web/servlet/controller	
service 层	com.atguigu.service	Service 接口包
	com.atguigu.service.impl	Service 接口实现类
dao 持久层	com.atguigu.dao	Dao 接口包
	com.atguigu.dao.impl	Dao 接口实现类
实体 bean 对象	com.atguigu.pojo/entity/domain/bean	JavaBean 类
测试包	com.atguigu.test/junit	
工具类	com.atguigu.utils	

搭建书城项目开发环境：



1、先创建书城需要的数据库和表。

```
drop database if exists book;

create database book;

use book;

create table t_user(
    `id` int primary key auto_increment,
    `username` varchar(20) not null unique,
    `password` varchar(32) not null,
    `email` varchar(200)
);

insert into t_user(`username`,`password`,`email`) values('admin','admin','admin@atguigu.com');

select * from t_user;
```

2、编写数据库表对应的 JavaBean 对象。

```
public class User {
    private Integer id;
    private String username;
    private String password;
    private String email;
```

3、编写工具类 JdbcUtils

3.1、导入需要的 jar 包（数据库和连接池需要）：

```
druid-1.1.9.jar  
mysql-connector-java-5.1.7-bin.jar
```

以下是测试需要：

```
hamcrest-core-1.3.jar  
junit-4.12.jar
```

3.2、在 src 源码目录下编写 jdbc.properties 属性配置文件：

```
username=root  
password=root  
url=jdbc:mysql://localhost:3306/book  
driverClassName=com.mysql.jdbc.Driver  
initialSize=5  
maxActive=10
```

3.3、编写 JdbcUtils 工具类：

```
public class JdbcUtils {  
  
    private static DruidDataSource dataSource;  
  
    static {  
        try {  
            Properties properties = new Properties();  
            // 读取 jdbc.properties 属性配置文件  
            InputStream inputStream =  
JdbcUtils.class.getClassLoader().getResourceAsStream("jdbc.properties");  
            // 从流中加载数据  
            properties.load(inputStream);  
            // 创建 数据库连接 池  
            dataSource = (DruidDataSource) DruidDataSourceFactory.createDataSource(properties);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
/**
 * 获取数据库连接池中的连接
 * @return 如果返回 null, 说明获取连接失败<br/>有值就是获取连接成功
 */
public static Connection getConnection(){

    Connection conn = null;

    try {
        conn = dataSource.getConnection();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return conn;
}

/**
 * 关闭连接, 放回数据库连接池
 * @param conn
 */
public static void close(Connection conn){
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}
```

3.4、JdbcUtils 测试

```
public class JdbcUtilsTest {

    @Test
    public void testJdbcUtils(){
        for (int i = 0; i < 100; i++){
            Connection connection = JdbcUtils.getConnection();
            System.out.println(connection);
            JdbcUtils.close(connection);
        }
    }
}
```

4、编写 BaseDao

4.1、导入 DBUtils 的 jar 包

commons-dbutils-1.3.jar

4.2、编写 BaseDao:

```
public abstract class BaseDao {

    //使用 DbUtils 操作数据库
    private QueryRunner queryRunner = new QueryRunner();

    /**
     * update() 方法用来执行: Insert\Update\Delete 语句
     *
     * @return 如果返回-1, 说明执行失败<br/>返回其他表示影响的行数
     */
    public int update(String sql, Object... args) {
        Connection connection = JdbcUtils.getConnection();
        try {
            return queryRunner.update(connection, sql, args);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.close(connection);
        }
        return -1;
    }

    /**
     * 查询返回一个 javaBean 的 sql 语句
     *
     * @param type 返回的对象类型
     * @param sql 执行的 sql 语句
     * @param args sql 对应的参数值
     * @param <T> 返回的类型的泛型
     * @return
     */
    public <T> T queryForOne(Class<T> type, String sql, Object... args) {
        Connection con = JdbcUtils.getConnection();
        try {
            return queryRunner.query(con, sql, new BeanHandler<T>(type), args);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JdbcUtils.close(con);
        }
    }
}
```

```
JdbcUtils.close(con);
}
return null;
}

/**
 * 查询返回多个 javaBean 的 sql 语句
 *
 * @param type 返回的对象类型
 * @param sql 执行的 sql 语句
 * @param args sql 对应的参数值
 * @param <T> 返回的类型的泛型
 * @return
 */
public <T> List<T> queryForList(Class<T> type, String sql, Object... args) {
    Connection con = JdbcUtils.getConnection();
    try {
        return queryRunner.query(con, sql, new BeanListHandler<T>(type), args);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        JdbcUtils.close(con);
    }
    return null;
}

/**
 * 执行返回一行一列的 sql 语句
 * @param sql 执行的 sql 语句
 * @param args sql 对应的参数值
 * @return
 */
public Object queryForSingleValue(String sql, Object... args){

    Connection conn = JdbcUtils.getConnection();

    try {
        return queryRunner.query(conn, sql, new ScalarHandler(), args);
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        JdbcUtils.close(conn);
    }
    return null;
}
}
```

5、编写 UserDao 和测试

UserDao 接口:

```
public interface UserDao {

    /**
     * 根据用户名查询用户信息
     * @param username 用户名
     * @return 如果返回 null, 说明没有这个用户。反之亦然
     */
    public User queryUserByUsername(String username);

    /**
     * 根据 用户名和密码查询用户信息
     * @param username
     * @param password
     * @return 如果返回 null, 说明用户名或密码错误, 反之亦然
     */
    public User queryUserByUsernameAndPassword(String username, String password);

    /**
     * 保存用户信息
     * @param user
     * @return 返回-1 表示操作失败, 其他是 sql 语句影响的行数
     */
    public int saveUser(User user);
}
```

UserDaoImpl 实现类:

```
public class UserDaoImpl extends BaseDao implements UserDao {

    @Override
    public User queryUserByUsername(String username) {
        String sql = "select `id`,`username`,`password`,`email` from t_user where username = ?";
        return queryForOne(User.class, sql, username);
    }

    @Override
    public User queryUserByUsernameAndPassword(String username, String password) {
        String sql = "select `id`,`username`,`password`,`email` from t_user where username = ? and password = ?";
        return queryForOne(User.class, sql, username, password);
    }

    @Override
```

```
public int saveUser(User user) {
    String sql = "insert into t_user(`username`,`password`,`email`) values(?,?,?)" ;
    return update(sql, user.getUsername(),user.getPassword(),user.getEmail());
}
```

UserDao 测试:

```
public class UserDaoTest {

    UserDao userDao = new UserDaoImpl();

    @Test
    public void queryUserByUsername() {

        if (userDao.queryUserByUsername("admin1234") == null ){
            System.out.println("用户名可用！");
        } else {
            System.out.println("用户名已存在！");
        }
    }

    @Test
    public void queryUserByUsernameAndPassword() {
        if ( userDao.queryUserByUsernameAndPassword("admin","admin1234") == null) {
            System.out.println("用户名或密码错误，登录失败");
        } else {
            System.out.println("查询成功");
        }
    }

    @Test
    public void saveUser() {
        System.out.println( userDao.saveUser(new User(null,"wzg168", "123456", "wzg168@qq.com")) );
    }
}
```

6、编写 UserService 和测试

UserService 接口:

```
public interface UserService {

    /**
     * 注册用户
     * @param user
     */
}
```



```
public void registUser(User user);

/**
 * 登录
 * @param user
 * @return 如果返回null, 说明登录失败, 返回有值, 是登录成功
 */
public User login(User user);

/**
 * 检查 用户名是否可用
 * @param username
 * @return 返回 true 表示用户名已存在, 返回 false 表示用户名可用
 */
public boolean existsUsername(String username);
}
```

UserServiceImpl 实现类:

```
public class UserServiceTest {

    UserService userService = new UserServiceImpl();

    @Test
    public void registUser() {
        userService.registUser(new User(null, "bbj168", "666666", "bbj168@qq.com"));
        userService.registUser(new User(null, "abc168", "666666", "abc168@qq.com"));
    }

    @Test
    public void login() {
        System.out.println( userService.login(new User(null, "wzg168", "123456", null)) );
    }

    @Test
    public void existsUsername() {
        if (userService.existsUsername("wzg16888")) {
            System.out.println("用户名已存在!");
        } else {
            System.out.println("用户名可用!");
        }
    }
}
```

UserService 测试:

```
public class UserServiceTest {

    UserService userService = new UserServiceImpl();
```

```
@Test
public void registUser() {
    userService.registUser(new User(null, "bbj168", "666666", "bbj168@qq.com"));
    userService.registUser(new User(null, "abc168", "666666", "abc168@qq.com"));
}

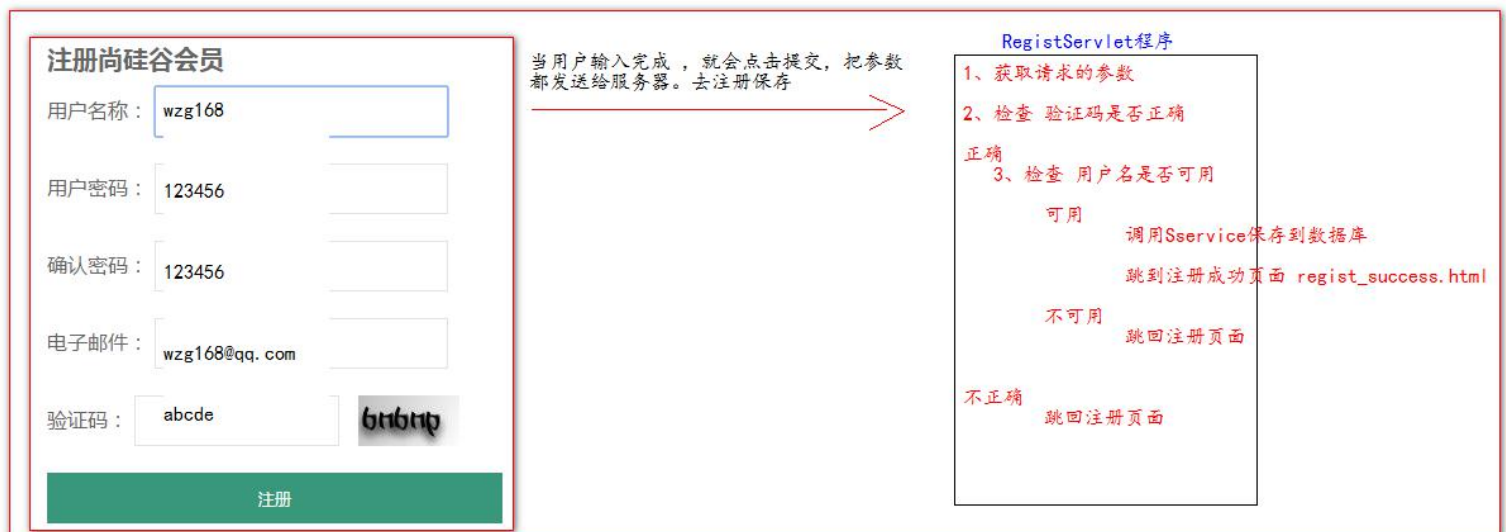
@Test
public void login() {
    System.out.println( userService.login(new User(null, "wzg168", "123456", null)) );
}

@Test
public void existsUsername() {
    if (userService.existsUsername("wzg16888")) {
        System.out.println("用户名已存在!");
    } else {
        System.out.println("用户名可用!");
    }
}
}
```

7、编写 web 层

7.1、实现用户注册的功能

7.1.1、图解用户注册的流程：



7.1.2、修改 regist.html 和 regist_success.html 页面

1、添加 base 标签

```
<!-- 写 base 标签，永远固定相对路径跳转的结果-->
<base href="http://localhost:8080/book/">
```

2、修改 base 标签对页面中所有相对路径的影响（浏览器 F12，哪个报红，改哪个）

以下是几个修改的示例：

```
<link type="text/css" rel="stylesheet" href="static/css/style.css" >
<script type="text/javascript" src="static/script/jquery-1.7.2.js"></script>
```

3、修改注册表单的提交地址和请求方式

```
</div>
<div class="form">
  <form action="registServlet" method="post">
    <label>用户名称: </label>
    <input class="itxt" type="text" placeholder="请输入用
```

7.1.3、编写 RegisterServlet 程序

```
public class RegisterServlet extends HttpServlet {

    private UserService userService = new UserServiceImpl();

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        // 1、获取请求的参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        String email = req.getParameter("email");
        String code = req.getParameter("code");

        // 2、检查 验证码是否正确 === 写死, 要求验证码为:abcde
        if ("abcde".equalsIgnoreCase(code)) {
            // 3、检查 用户名是否可用
            if (userService.existsUsername(username)) {
                System.out.println("用户名[" + username + "]已存在!");
                // 跳回注册页面
                req.getRequestDispatcher("/pages/user/regist.html").forward(req, resp);
            }
        }
    }
}
```

```

    } else {
        //      可用
        // 调用 Sservice 保存到数据库
        userService.registUser(new User(null, username, password, email));

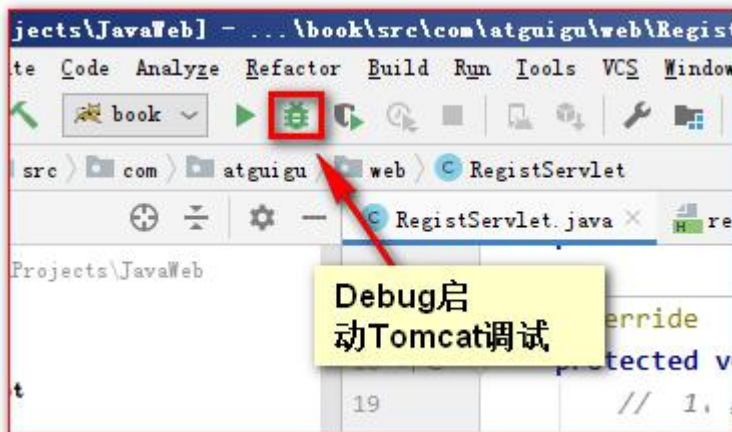
        //      跳到注册成功页面 regist_success.html
        req.getRequestDispatcher("/pages/user/regist_success.html").forward(req, resp);
    }
} else {
    System.out.println("验证码[" + code + "]错误");
    req.getRequestDispatcher("/pages/user/regist.html").forward(req, resp);
}
}
}

```

7.2、IDEA 中 Debug 调试的使用

7.2.1、Debug 调试代码，首先需要两个元素：断点 + Debug 启动服务器

- 1、断点，只需要在代码需要停的行的左边上单击，就可以添加和取消
- 2、Debug 启动 Tomcat 运行代码：



7.2.2、测试工具栏：



让代码往下执行一行。



可以进入当前方法体内（自己写的代码，非框架源码）



跳出当前方法体外



强制进入当前方法体内



停在光标所在行（相当于临时断点）

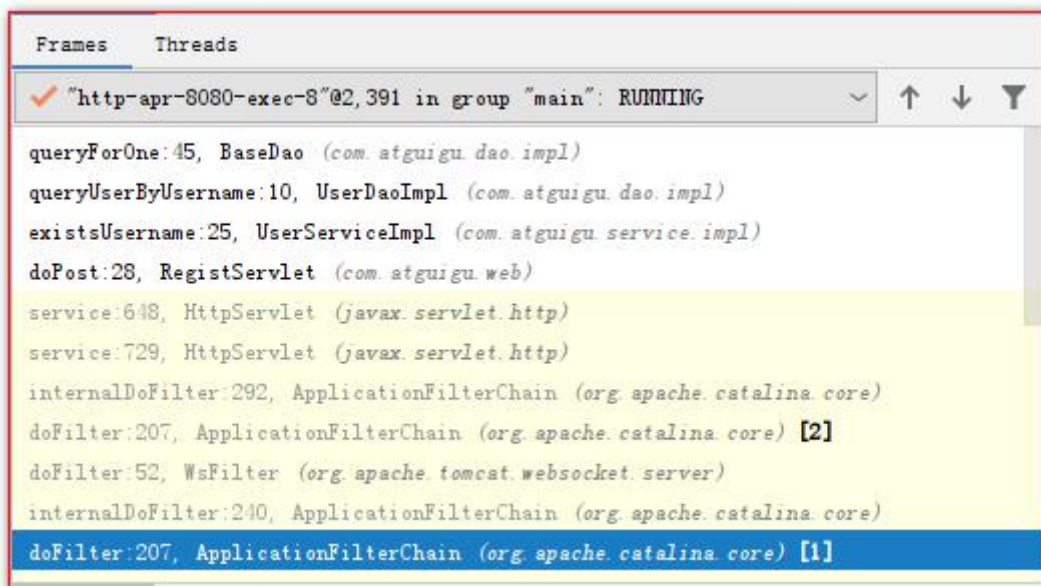
7.2.3、变量窗口

变量窗口：它可以查看当前方法范围内所有有效的变量。

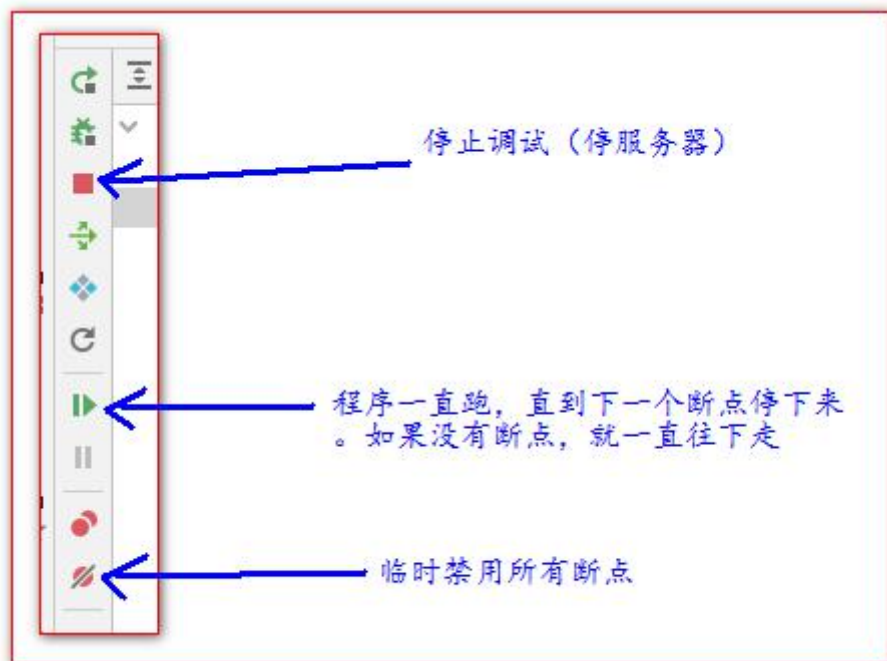


7.2.4、方法调用栈窗口

- 1、方法调用栈可以查看当前线程有哪些方法调用信息
- 2、下面的调用上一行的方法

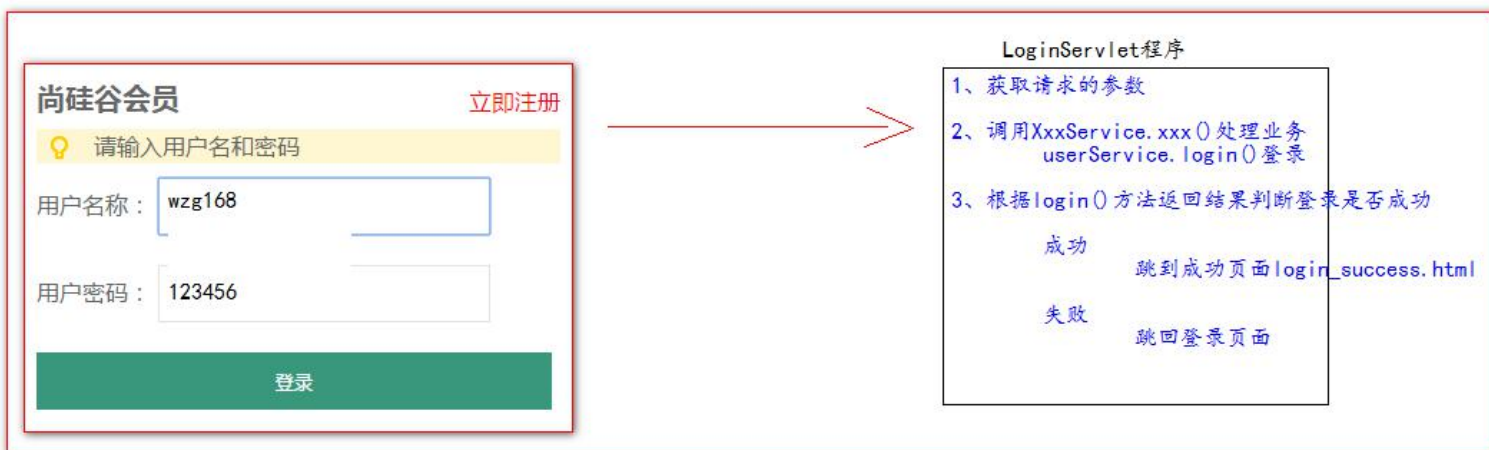


其他常用调试相关按钮：



7.3、用户登录功能的实现

7.3.1、图解用户登录



7.3.2、修改 login.html 页面和 login_success.html 页面

1、添加 base 标签

```
<!-- 写 base 标签，永远固定相对路径跳转的结果 -->
<base href="http://localhost:8080/book/">
```


2、修改 base 标签对页面中所有相对路径的影响（浏览器 F12，哪个报红，改哪个）

以下是几个修改的示例：

```
<link type="text/css" rel="stylesheet" href="static/css/style.css" >
<script type="text/javascript" src="static/script/jquery-1.7.2.js"></script>
```

3、修改 login.html 表单的提交地址和请求方式

```
<span class= errorMSG >请输入用户名和密码</span>
</div>
<div class="form">
  <form action="loginServlet" method="post">
    <label>用户名称: </label>
    <input class="itxt" type="text" placeholder="请输入用户名"
      autocomplete="off" tabindex="1" name="username" />
```

7.3.3、LoginServlet 程序

```
public class LoginServlet extends HttpServlet {

    private UserService userService = new UserServiceImpl();

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        // 1、获取请求的参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        // 调用 userService.login() 登录处理业务
        User loginUser = userService.login(new User(null, username, password, null));
        // 如果等于 null, 说明登录 失败!
        if (loginUser == null) {
            // 跳回登录页面
            req.getRequestDispatcher("/pages/user/login.html").forward(req, resp);
        } else {
            // 登录 成功
            // 跳到成功页面 login_success.html
            req.getRequestDispatcher("/pages/user/login_success.html").forward(req, resp);
        }
    }
}
```

2.项目阶段二：用户注册和登陆的实现。

需求 1：用户注册

需求如下：

- 1) 访问注册页面
- 2) 填写注册信息，提交给服务器
- 3) 服务器应该保存用户
- 4) 当用户已经存在----提示用户注册 失败，用户名已存在
- 5) 当用户不存在-----注册成功

需求 2：用户登陆

需求如下：

- 1) 访问登陆页面
- 2) 填写用户名密码后提交
- 3) 服务器判断用户是否存在
- 4) 如果登陆失败 --->>>> 返回用户名或者密码错误信息
- 5) 如果登录成功 --->>>> 返回登陆成功 信息

