

最近因为项目原因在挖洞。

其实已经有段时间没挖洞了，java 都有点忘记了。

这里想整体复习一下，顺便记录下学习流程。

学习的中心思想就是最短时间学完最多的东西，就是要快。

慢 != 扎实

学了很多遍+做了练习 = 扎实

如果学一个东西感觉很吃力不是自己的原因，是因为很多前置知识没有掌握，这个时候应该先把简单的前置知识掌握了，学起来会好很多。

线性思维我实操下来真的不好使，学到后面，前面的东西又忘记了。

环形思维好用很多。

不求一次完美，用一次次 **looper** 来加深理解，最后效果很好。

我这个快速学习，经过我自己实践证明，是很好用的。

因为我大学看高数考前突击也就是三天时间（基本没去上过课），其他简单的课也就是一晚上的时间（基本没上过课），最后分数虽然没有很高，但是都过了。

但快速学习也不是瞎学，有以下要点

1、针对实战培养感觉：

也就是说，先直面最终需要解决的实际问题，然后再反哺回来，看自己需要哪些知识，切忌一上来就对着基础知识一个个啃，效率非常之低下，而且没有感觉，因为学了不知道有啥用。

就像大学老师划重点，我就看他画的重点，所以才能学的那么快，而且也能过关。

28 法则在哪里都适用，有些东西用的少，就少看，用的多，就多看，眉毛胡子一把抓，势必分散精力，最后效果不好。

2、不会的问题不纠结

不会不是我智商有问题，而是我的方法有问题。

记住这句话，问题先解决 **99%**。

如果世界上的问题都可以用死磕来解决，那么也不需要什么方法了。

我很不喜欢那些误导人的文章和言论，说一些什么鬼话，什么天天努力到吐啥的，什么遇到难题坚持不懈，什么头昏脑胀还坚持学习。

都他妈放屁的，努力到吐，说明过载了。

机器过载会影响使用寿命，人也是一样，可持续发展才是硬道理。

坚持不懈，中等，简单难度的东西还行，遇到很难的问题你试试，你再怎么坚持也不可能解出来，因为少了前置的东西。

训练大脑就像训练肌肉一样，讲究循序渐进。

比如举重运动，就算是奥运会冠军，都是从 **30kg**，**40kg**，**50kg** 这样一点点累加上来的，没有人没经过训练一上来就是 **100kg**，**200kg**，冠军也不行，也要遵守基本方法。

大脑也是一样，只能学习稍微超出当前范围一点点的知识，这个范围称为学习区，超出了，大脑就当机，当机的表现就是昏昏欲睡，注意力分散，这是基本规律，不信可以试试。

3、注意劳逸结合

劳逸结合不是一句空话，而是一句实在话。

就像健身一样，跑步累了，就要休息，睡一觉，然后第二天再继续训练，或者第三天。

学习也是一样，只不过跑步锻炼的是身体的肌肉，学习锻炼的是大脑的肌肉。

学累了，就要休息，有时候不一定是睡觉，就像跑步累了不一定是睡觉一样。

可以打打游戏，玩玩手机之类的，放松一下，然后过一会其实感觉又可以了，那就继续学。

以上三点，道理都很简单，但是世界上偏偏充满了很多不希望别人学好的人，于是放出了大量的毒鸡汤，过分强调意志力还有一些主观层面的东西的重要性，是真的影响别人学习。

意志力是什么？比如一个人举重，平常只能举起 100kg，然后现在参加比赛，面对这么多观众，心中激荡，于是决定挑战自我，举 110kg，最后成功了。

意志力的作用，更多是面对极限环境的一种自我激素调动，然后让自己的能力能够比平时超出一些范围，但不多。

而且超出极限，还有副作用，需要更多的休息。

超出那一点点极限，然后浪费我更多的时间来休息，我认为不划算。

因此我除了比赛的时候，平时基本用不到意志力，就是遵循客观规律，规划时间，学习生活也没耽误，还有时间打游戏。

学习能力是贯穿人的一生的，是我认为最值得培养的一种能力，没有之一。

而且一通百通，这套方法其实放在任何学科都适用。

不要和身体产生的事实进行抗辩，身体饿了，就去吃饭，身体困了，就睡觉，身体注意力分散，就去玩玩游戏或者看看视频，要遵循身体基本的物理规律，因为身体的底层代码不是我们自己写的，我们没法改，只能遵循他写好的规律，我们去用，就像我们使用任何工具一样，遵循当前工具本身的使用规律，就能取得事半功倍的效果。

遵循规律，利用规律，最后组合创新规律，最后拥有体检更佳的人生之旅，真的不试一试嘛。

以下是我自己的 java 学习实例，仅作为参考。

---java 部分学习记录---

我 java 不是零基础，但我自己感觉跟零基础也差不多>_<，所以感觉这个流程，大家初入门，或者如果很久没看 java 的人，应该也可以用。

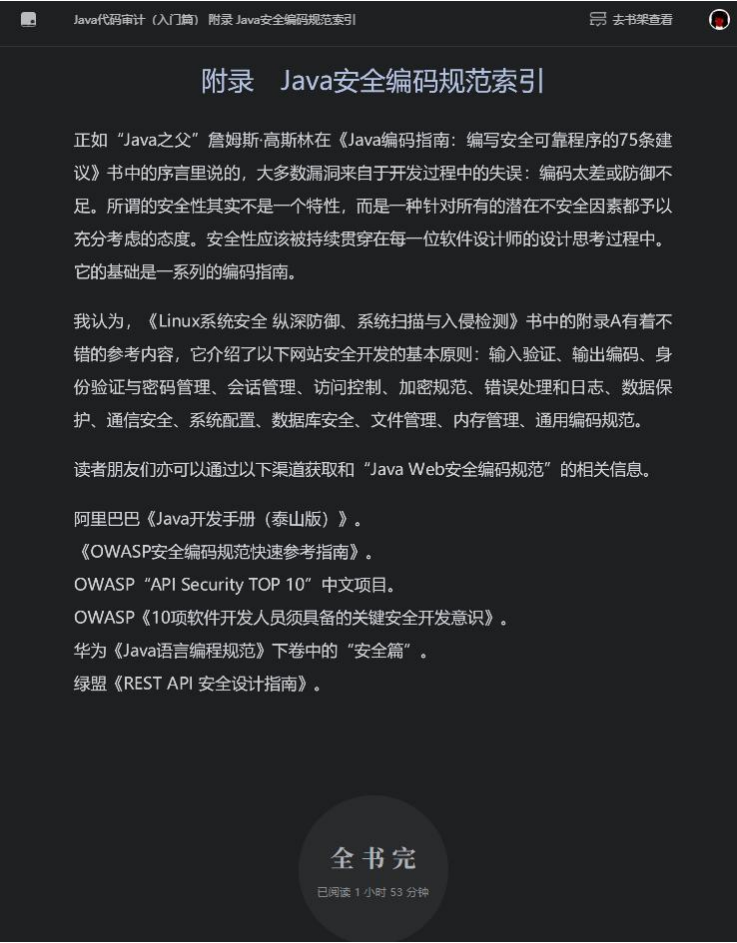
预计用 5-7 天时间，想把整个 java 的基础+web+框架整个过一遍。

下面边写会边列出时间线

---20220505 12:00---

1、全局观建立

很快的时间过了一遍 java 代码审计这本书



我用的是微信读书看的

东西不用都细看，反正我就看 rce 相关的东西。

比如



5.6 安全配置错误

5.6.1 安全配置错误漏洞简介

5.6.2 Tomcat任意文件写入 (CVE-2017-12615)

5.6.3 Tomcat AJP 文件包含漏洞 (CVE-2020-1938)

5.6.4 Spring Boot远程命令执行

5.6.5 小结

5.8 不安全的反序列化

5.8.1 不安全的反序列化漏洞简介

5.8.2 反序列化基础

5.8.3 漏洞产生的必要条件

5.8.4 反序列化拓展

5.8.5 Apache Commons Collections反序列化漏洞

5.8.6 FastJson反序列化漏洞

5.8.7 小结

5.9 使用含有已知漏洞的组件

5.9.1 组件漏洞简介

5.9.2 Weblogic中组件的漏洞

5.9.3 富文本编辑器漏洞

5.9.4 小结

第7章 Java EE开发框架安全审计
7.1 开发框架审计技巧简介
7.1.1 SSM框架审计技巧
7.1.2 SSH框架审计技巧
7.1.3 Spring Boot框架审计技巧
7.2 开发框架使用不当范例 (Struts2 远程代码执行)
7.2.1 OGNL简介
7.2.2 S2-001漏洞原理分析

8.5 单点漏洞审计
8.5.1 SQL审计
8.5.2 XSS 审计
8.5.3 SSRF审计
8.5.4 RCE审计

然后心中大概有数了

不要求完全记下来，但是需要了解大概 java 代码审计是要干什么

---20220505 14:44---

2、做题

先做一套题练练手，那么这里选择书中举例的 jspxcms

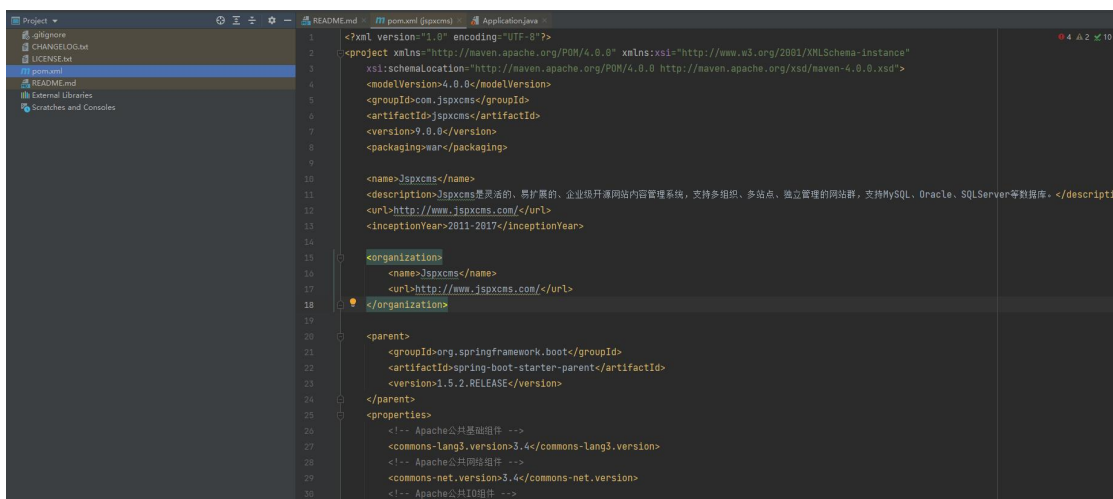
初期做题一定要选有答案的题，不然很容易打击信心。

先知上也有答案可以参考

//<https://xz.aliyun.com/t/10891>

然后发现一个问题

比如第一步，环境搭建

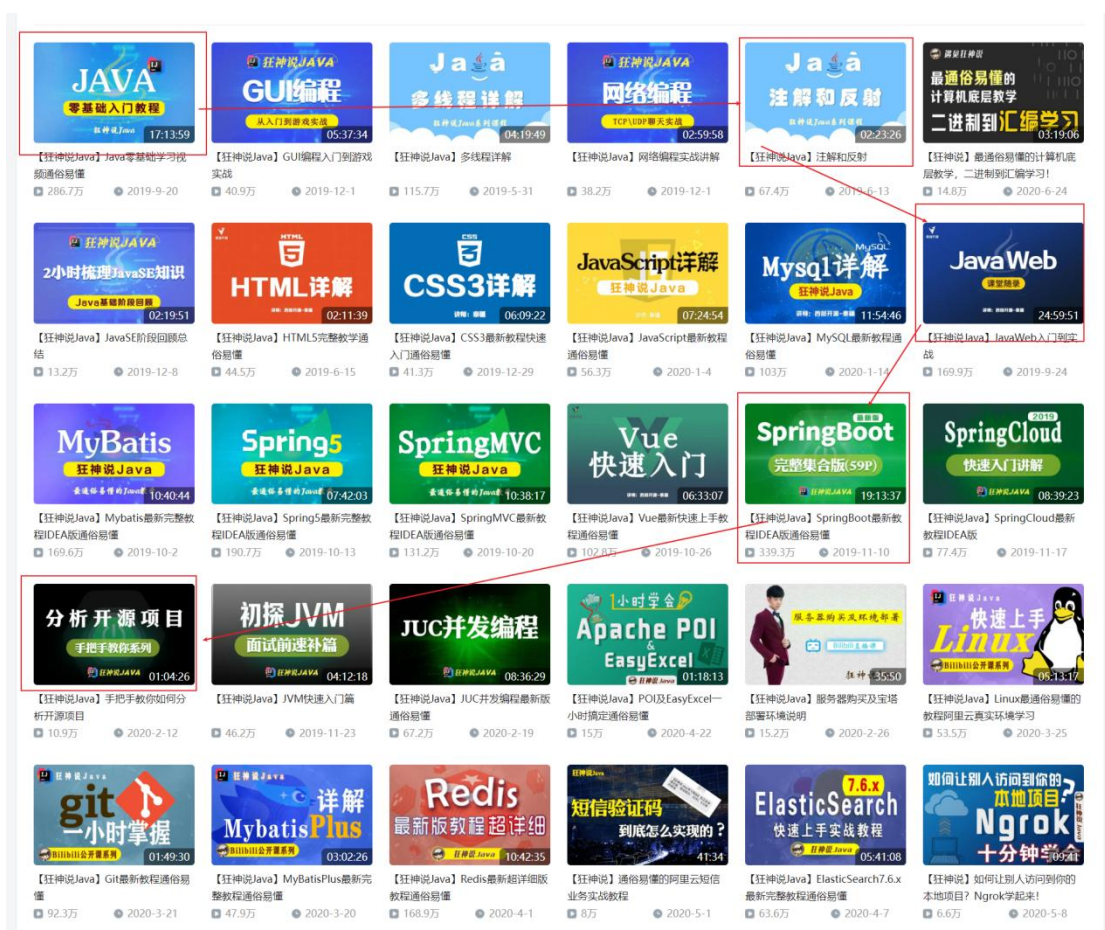


东西我都搞好了，但是 springboot 跑不起来
这里排错出现了问题，得去看 java 开发基础。

推荐 b 站看视频+看文字教程结合

视频的好处在于他的东西你都能复现出来

文字教程的好处在于速度快，但是他写的东西你不一定能够都复现出来，因为时代的原因或者因为作者省略了一些步骤。



打算按照上面红线的顺序过一遍

---20220505 20:02---

睡了一觉，然后吃了个饭，回来就七点了。

Springboot 的东西看了七小节

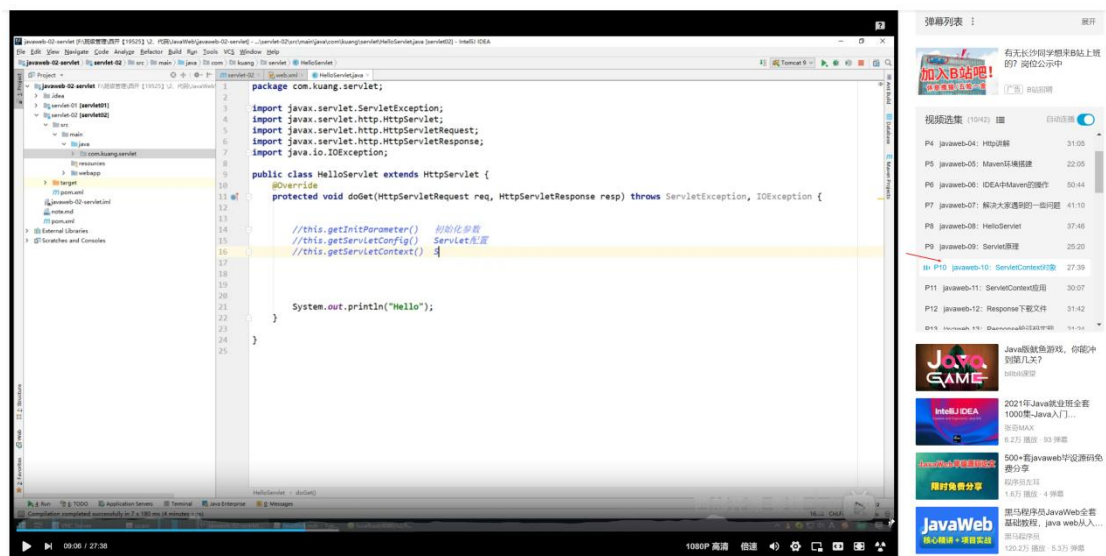
视频选集 (7/61)			自动连播
P1	1、这个阶段该如何学习	16:07	
P2	2、什么是SpringBoot	16:46	
P3	3、什么是微服务架构	16:44	
P4	4、第一个springboot程序	21:41	
P5	5、IDEA快速创建及彩蛋	10:02	
P6	6、Springboot自动装配原理	43:20	
P7	7、了解下主启动类怎么运行	12:20	

看不下去了，因为没应用场景，而且少了前置知识

跳到 javaweb 那边去

会的我就不看了

直接查漏补缺，看不会的



从 servletContext 相关开始看起

配置 context

```
package com.fuck.servlet;

import ...

public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        String username = "fucku";
        context.setAttribute("username", username);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        super.doPost(req, resp);
    }
}
```

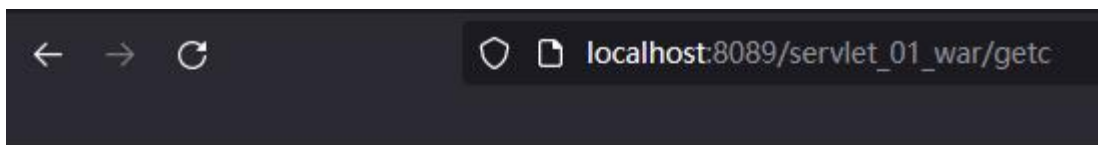

Web.xml 定义

```
<servlet>
  <servlet-name>fuck</servlet-name>
  <servlet-class>com.fuck.servet.GetServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>fuck</servlet-name>
  <url-pattern>/fuck</url-pattern>
</servlet-mapping>

</web-app>
```

取出 context



name:fucku

看的时候多用>键来快进，会的就不看了，别耽误时间。

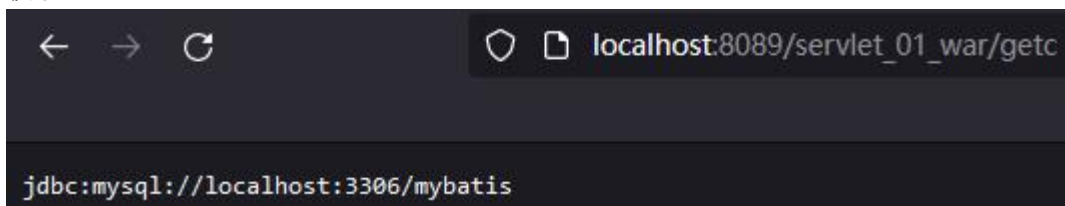
应用 1

获取 param

配置好 web.xml

```
<context-param>
  <param-name>url</param-name>
  <param-value>jdbc:mysql://localhost:3306/mybatis</param-value>
</context-param>
```

获取



Context 理解

就是一个应用中的一个公共文件夹，可以往这里面放东西，然后大家都可以用。

Context 转发：

转发器

```
public class DispatchServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        RequestDispatcher requestDispatcher = context.getRequestDispatcher("/fuck");
        requestDispatcher.forward(req, resp);
    }

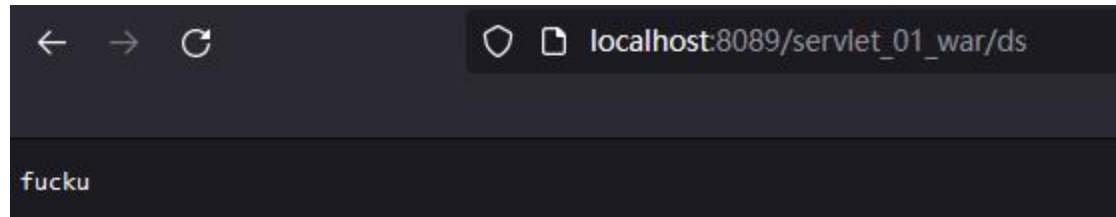
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        super.doPost(req, resp);
    }
}
```

被转发的输出单元

```
public class FuckServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        resp.getWriter().print("fucku");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        super.doPost(req, resp);
    }
}
```

结果



转发和重定向

转发 a-b-c

重定向 a-b b-a a-c

文件下载

```
public class FileDownload extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //获取文件下载路径
        String realPath = "java\\projectnormal\\servlet-01\\target\\classes\\fuck.png";
        System.out.println("path:" + realPath);
        //下载的文件name是啥
        String filename = realPath.substring(realPath.lastIndexOf("\\") + 1);
        //想办法让浏览器支持我们需要下载的东西
        resp.setHeader("Content-Disposition", "attachment;filename=" + filename);
        //获取下载输入
        FileInputStream in = new FileInputStream(realPath);
        //创建缓冲区
        int len = 0;
        byte[] buffer = new byte[1024];
        //获取outputstream
        ServletOutputStream out = resp.getOutputStream();
        //将fileoutputstream写入buffer
        while((len = in.read(buffer)) > 0){
            out.write(buffer, 0, len);
        }
    }
}
```

结果



验证码

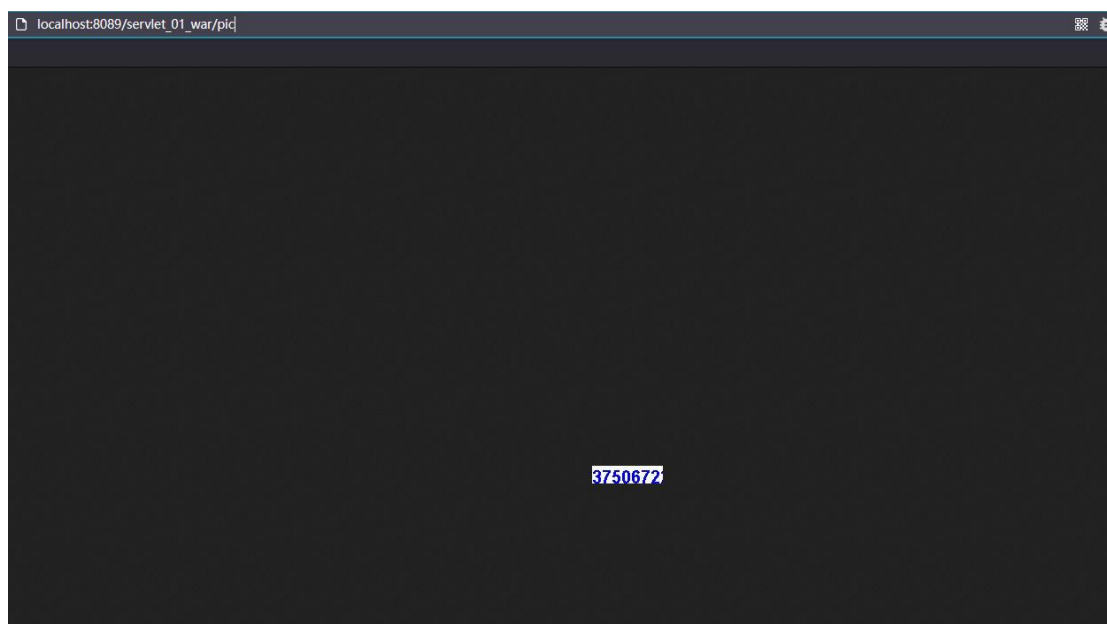
```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    //浏览器3s刷新一次
    resp.setHeader("name: refresh", "value: 3");
    //定义一个图片
    BufferedImage image = new BufferedImage(width: 80, height: 20, BufferedImage.TYPE_INT_RGB);
    //画笔
    Graphics2D g = (Graphics2D) image.getGraphics();
    //设置背景颜色
    g.setColor(Color.white);
    g.fillRect(x: 0, y: 0, width: 80, height: 20);
    //写入

    g.setColor(Color.BLUE);
    g.setFont(new Font("name: null, Font.BOLD, size: 20));
    g.drawString(makeNum(), x: 0, y: 20);

    //告诉浏览器这个请求是图片
    resp.setContentType("image/jpeg");
    //清除浏览器缓存
    resp.setDateHeader("name: expires", "date: -1");
    resp.setHeader("name: Cache-Control", "value: no-cache");
    resp.setHeader("name: Pragma", "value: no-cache");

    //传给浏览器
    boolean write = ImageIO.write(image, formatName: "jpg", resp.getOutputStream());
}

private String makeNum(){
    Random random = new Random();
    String num = random.nextInt(bound: 99999999) + "";
    StringBuffer sb = new StringBuffer();
    for(int i = 0; i < 7 - num.length(); i++){
        sb.append("0");
    }
    num = sb.toString() + num;
}
```



这里觉得他写的这个验证码算法有点意思，做了个小实验

```

public class Test11 {

    public static void main(String[] args) {
        Random random = new Random();
        String num = random.nextInt( bound: 9999) + "";

        System.out.println(num);
        System.out.println(num.length());
        StringBuffer sb = new StringBuffer();
        for(int i = 0; i < 7 - num.length(); i++){
            sb.append("0");//相当于位数不够就在前面补0
        }
        num = sb.toString() + num;
        System.out.println(num);
    }
}

```

结果

```

"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
6689
4
0006689

```

重定向

B一个web资源收到客户端A请求后，B他会通知A客户端去访问另外一个web资源C，这个过程叫重定向

和转发有区别

很简单

代码：

```

public class RedirectServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        resp.sendRedirect( location: "/servlet_01_war/fuck");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        super.doPost(req, resp);
    }
}

```

抓包：

Request

PrettyRaw\nActions

1GET /servlet_01_war/redict HTTP/1.1

2Host: localhost:8089

3User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0

4Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

5Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

6Accept-Encoding: gzip, deflate

7Connection: close

8Cookie: JSESSIONID=D715A742D22EAD30360C8E74C0D7DB6D; Idea-7f7ed34b=37b0a42b-b86c-4f53-88b6-097493e07446; JSESSIONID=54D6F39E5BE50F54B845954D7C2C0DED

9Upgrade-Insecure-Requests: 1

10Sec-Fetch-Dest: document

11Sec-Fetch-Mode: navigate

12Sec-Fetch-Site: none

13Sec-Fetch-User: ?1

14

Response

PrettyRawRender\nActions

1HTTP/1.1 302

2Location: /servlet_01_war/fuck

3Content-Length: 0

4Date: Thu, 05 May 2022 14:27:56 GMT

5Connection: close

6

7

结果:

Request

PrettyRaw\nActions

1GET /servlet_01_war/fuck HTTP/1.1

2Host: localhost:8089

3User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0

4Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8

5Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

6Accept-Encoding: gzip, deflate

7Connection: close

8Cookie: JSESSIONID=D715A742D22EAD30360C8E74C0D7DB6D; Idea-7f7ed34b=37b0a42b-b86c-4f53-88b6-097493e07446; JSESSIONID=54D6F39E5BE50F54B845954D7C2C0DED

9Upgrade-Insecure-Requests: 1

10Sec-Fetch-Dest: document

11Sec-Fetch-Mode: navigate

12Sec-Fetch-Site: none

13Sec-Fetch-User: ?1

14

Response

PrettyRawRender\nActions

1HTTP/1.1 200

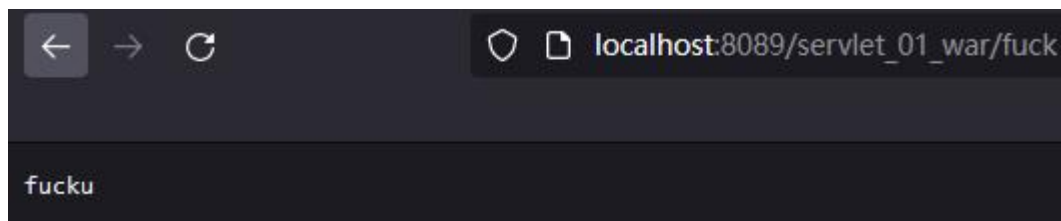
2Content-Length: 5

3Date: Thu, 05 May 2022 14:28:13 GMT

4Connection: close

5

6fucku



HttpRequest

写了两行

```

public class RequestTest extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        req.setCharacterEncoding("utf-8");
        String username = req.getParameter( name: "username");
        String password = req.getParameter( name: "password");

        System.out.println(username + ":" + password);

    }
}

```

结果:

```

05-May-2022 23:02:55.071 淇℃佷
05-May-2022 23:02:55.110 淇℃佷
fucku:fucku

```

简单

XD

Cookie

就是 ticket

很多都是票据逻辑

电影票凭票入场等等

kerberos 也是

代码

```

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    req.setCharacterEncoding("utf-8");
    resp.setCharacterEncoding("utf-8");

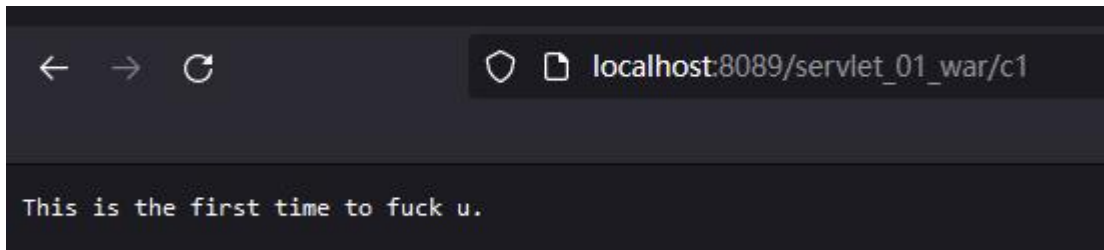
    PrintWriter out = resp.getWriter();
    Cookie[] cookies = req.getCookies();

    if (cookies != null){
        //取出cookie
        for(int i = 0; i < cookies.length; i++){
            Cookie cookie = cookies[i];
            if(cookie.getName().equals("LastLoginTime")){
                long lastLoginTime = Long.parseLong(cookie.getValue());
                Date date = new Date(lastLoginTime);
                out.write(date.toLocaleString());
            }
        }
    }
    else{
        out.write( s: "This is the first time to fuck u.");
    }
    Cookie cookie = new Cookie( name: "LastLoginTime", value: System.currentTimeMillis() + "");
    resp.addCookie(cookie);
}

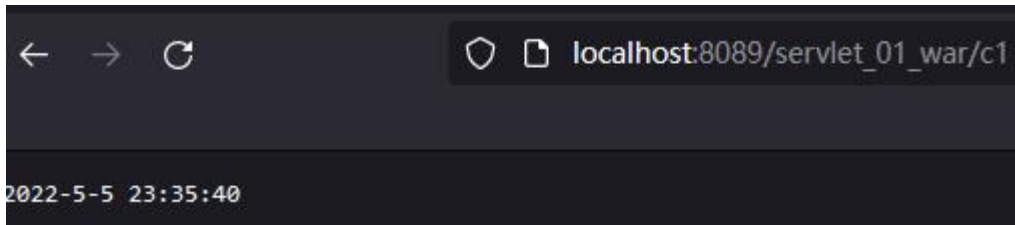
```

结果

第一次访问



第二次访问



---20220506 14: 41---

打了一通宵游戏，睡到刚刚才起

Session

创建 session

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException {
    HttpSession session = req.getSession();
    session.setAttribute( name: "name", value: "fuck");
    String sessionId = session.getId();

    if(session.isNew()){
        resp.getWriter().write( s: "session created:" + sessionId);
    }else{
        resp.getWriter().write( s: "session already exists" + sessionId);
    }
}
```

读取 session

```
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException {
    req.setCharacterEncoding("UTF-8");
    resp.setCharacterEncoding("UTF-8");
    resp.setContentType("text/html;charset=utf-8");

    HttpSession session = req.getSession();

    String session1 = (String) session.getAttribute( name: "name");
    resp.getWriter().print(session1);
}
```

就理解为字符串的增删改查就行

流程

第一次请求 给 cookie

第二次请求 带上 cookie cookie 中有 sessionid，服务端用 sessionid 来区分用户是谁

Jsp 原理剖析

Jsp 看起来是前端技术，其实是后端技术

就是在 jsp 中用 java 的格式来写 java

Jsp 基础语法

看看即可

到时候写的时候再查

内置对象和作用域

看看即可

Jstl 标签

一种简化的 jsp 语法

写马过 waf 可能有用

先了解即可

JavaBean

和 jsp 联动的一个东西

JavaBean 理解为一个公共类

大家都可以用

然后就不用重复写这个类了

写在 pojo 里就行

MVC

M model

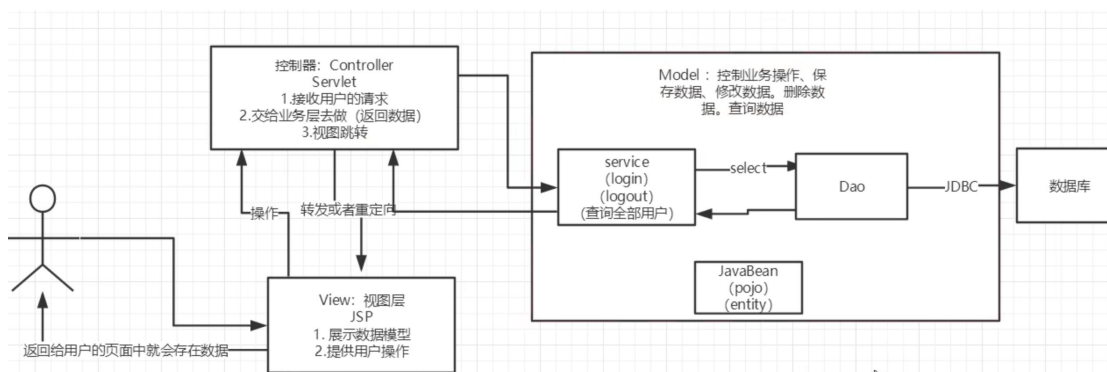
V view

C controller

早期架构：

Servlet 中写了 crud（增删改查），也就是说 servlet 中有具体逻辑，高耦合，不便维护

演进架构：



他画的很清楚了

相当于业务逻辑 数据库 实体类 各自独立

然后 javaBean 作为公共实体类单独分出来

Service 写具体的业务逻辑

Dao 层写 select 之类的查询语句去跟数据库交互

低耦合

在 springcloud 之前都是这个架构

Filter //内存马可用

定义一个 filter 类

```
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
    request.setCharacterEncoding("utf-8");
    response.setCharacterEncoding("utf-8");
    response.setContentType("text/html;charset=UTF-8");

    System.out.printf("fucku");
    chain.doFilter(request, response); // 转接请求
}
```

配置路由

```
<filter>
    <filter-name>Filter111</filter-name>
    <filter-class>com.fuck.servlet.FilterDemo1</filter-class>
</filter>

<filter-mapping>
    <filter-name>Filter111</filter-name>
    <url-pattern>/aaa</url-pattern>
</filter-mapping>
```

然后访问



回显

```
fuckufuckufuckufuckufucku
```

Listener 监听器//内存马可用

像个地雷一样，一碰开关就会爆炸

比如这个 sessionCreated

如果创建了 session 就会触发这个机制

```
public void sessionCreated(HttpSessionEvent se) {
```

代码懒得全写了

利用 filter 来实现权限拦截

放一个 filter 放到/servlet/*下面

然后判断 session

如果 session 为空

直接跳转到别的页面

Session

Jdbc 事务

要么都成功

要么都失败

ACID 原则

---20220507 0:33---

晚上十点多睡了一觉 现在醒来了

smbms 项目搭建

//通过自己做一个项目来熟悉一个项目的框架，将来方便审计

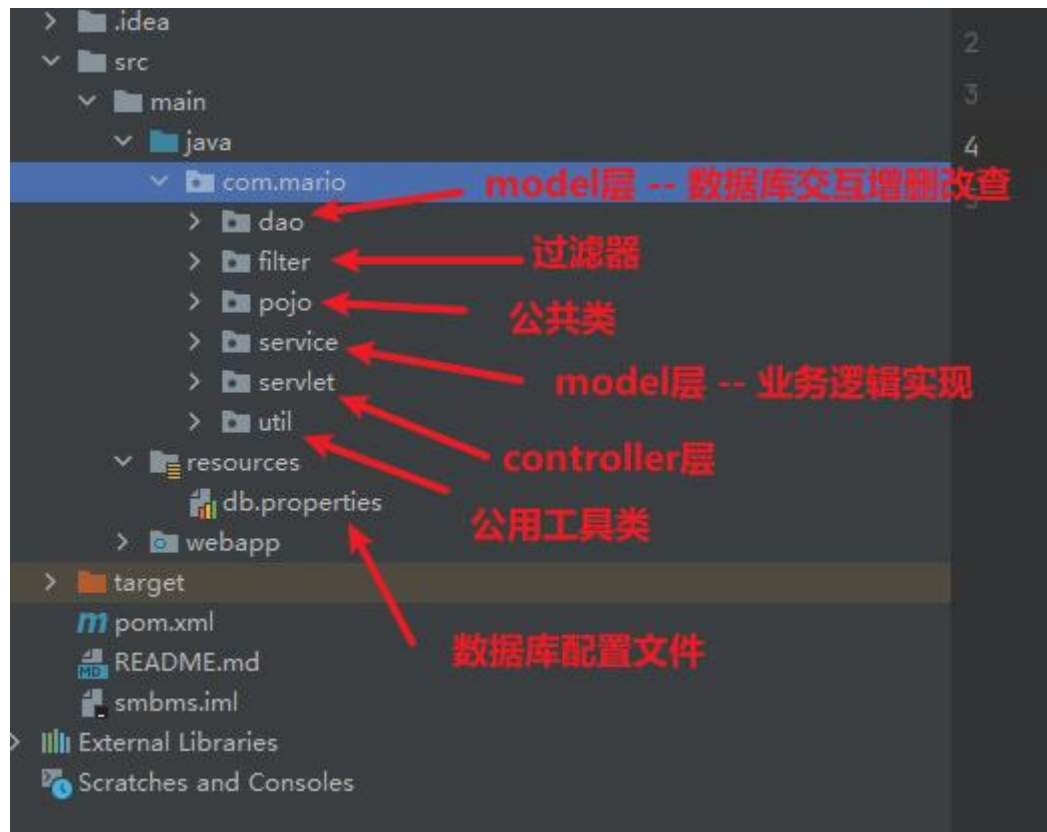
//整完这个项目 基础的 javaweb 估计就差不多了

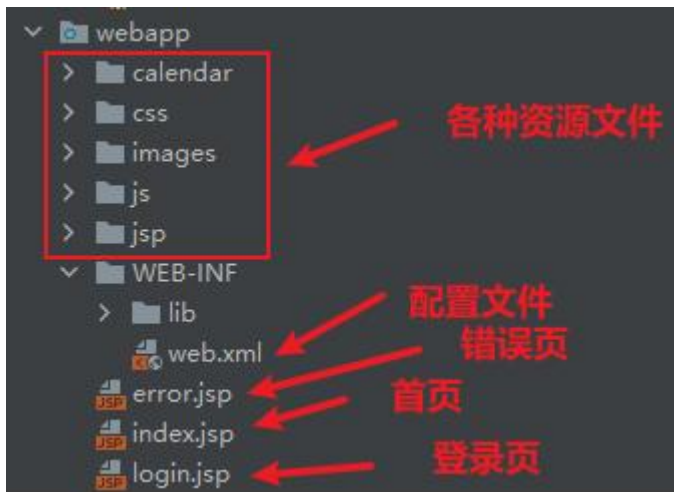
//后续也可以根据这个架构演进使用 spring mvc springboot 等

为了节约时间 这里代码就不手打了

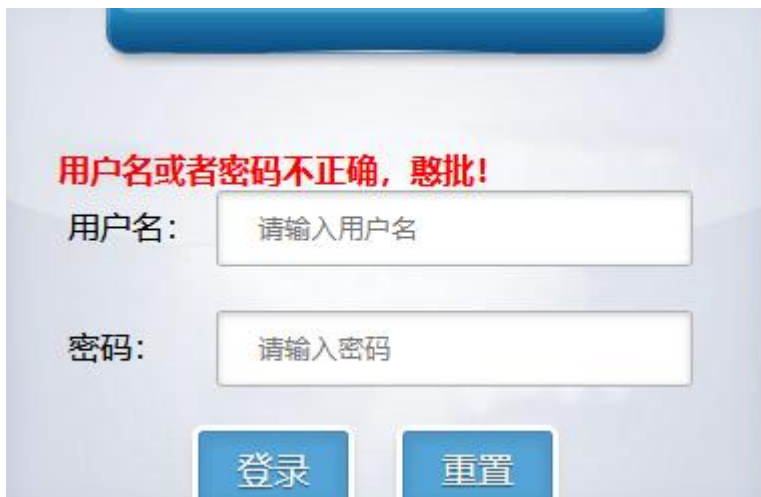
直接下了一套一样的

老规矩 先看架构





先从前端页面看起
死活登不进去



排错

```
java.sql.SQLException: Create breakpoint : The server time zone value '?N???????' is unrecognized or represents
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:129)
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:97)
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:89)
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:63)
    at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:73)
```

说老子时区有问题 妈的
Jdbc 加上一行连接

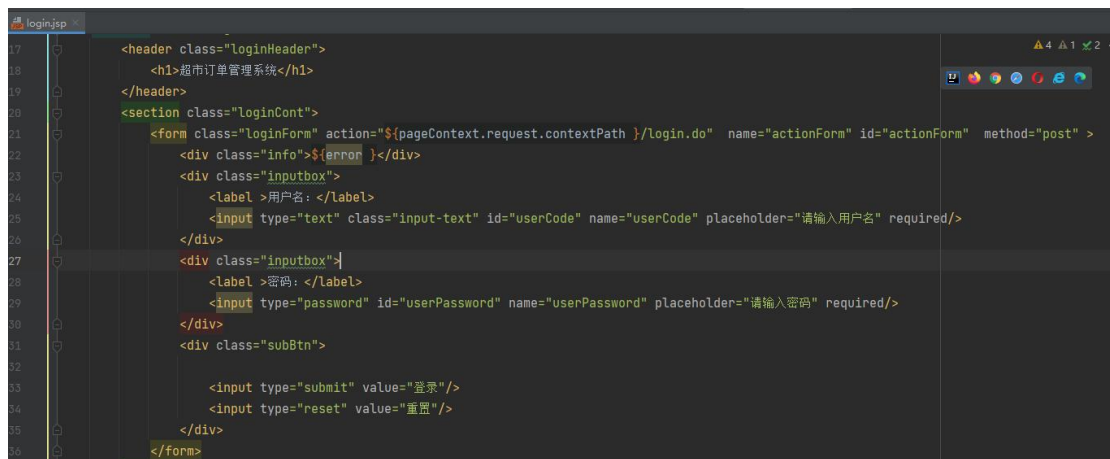
```
SL=true&serverTimezone=GMT%2B8
```

可以了

```
LoginServlet--start....
wen
123
07-Nov-2022 01:51:00
```



一个比较简单的系统
源码从前端开始看起
一个登录页面



传参往

```
<form class="loginForm" action="${pageContext.request.contextPath }/login.do" name="actionForm" id="actionForm" method="post" >
```

这里传

看 web.xml 其中的 servlet

```

<!-- 注册登录页面的Servlet-->
<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.mario.servlet.user.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login.do</url-pattern>
</servlet-mapping>
<!-- 注册注销页面-->

```

定位具体的 servlet 类

```

@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

    System.out.println("LoginServlet--start...");
    //获取用户名和密码
    String userCode = req.getParameter( name: "UserCode");
    String userPassword = req.getParameter( name: "UserPassword");
    //和数据库中的密码进行对比，调用业务层；
    UserService userService = new UserServiceImpl();
    User user = userService.login(userCode, userPassword); //这里已经把登录的人给查出来了
    System.out.println(userCode);
    System.out.println(userPassword);
    if (user!=null){ //查有此人，可以登录
        //将用户的信息放到Session中；
        req.getSession().setAttribute(Constants.USER_SESSION,user);
        //跳转到主页重定向
        resp.sendRedirect( location: "jsp/frame.jsp");
    }else { //查无此人，无法登录
        //转发回登录页面，顺带提示它，用户名或者密码错误；
        req.setAttribute( name: "error", o: "用户名或者密码不正确，憋批！");
        req.getRequestDispatcher( path: "login.jsp").forward(req,resp);
    }
}

```

这里由 view 层转到 controller 层

然后这里的具体查询登录的人的账号密码

Controller 没有具体的业务逻辑

业务逻辑在 service 里面完成

```

UserService userService = new UserServiceImpl();
User user = userService.login(userCode, userPassword); //这里已经把登录的人给查出来了

```

下个断点调试下 直观一些

The screenshot shows an IDE with a code editor and a debugger window. The code editor displays the login method of LoginServlet, with a breakpoint set at the line `User user = userService.login(userCode, userPassword);`. The debugger window shows the state of variables during execution. The variables are:

- `this`: (LoginServlet@3452)
- `req`: (RequestFacade@3447)
- `resp`: (ResponseFacade@3448)
- `userCode`: "123"
- `userPassword`: "123"
- `userService`: (UserServiceImpl@3451)

The variables `userCode` and `userPassword` are highlighted with a red box, indicating they are the values being passed to the login method.

Into 进来看一下

```
gin.jsp x web.xml x LoginServlet.java x UserServiceImpl.java x
public User login(String userCode, String password) {    userCode: "123"    password: "123"
    Connection connection = null;
    User user = null;

    try {
        connection = BaseDao.getConnection();
        //通过业务层调用对应的具体的数据库操作
        user = userDao.getLoginUser(connection, userCode);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        BaseDao.closeResource(connection, preparedStatement: null, resultSet: null);
    }

    //匹配密码
    if(null != user){
        if(!user.getUserPassword().equals(password))
            user = null;
    }

    return user;
}
```

还是没有具体的数据库查询语句

只是做了逻辑匹配

再 into 进去看看

```
public User getLoginUser(Connection connection, String userCode) throws SQLException {
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    User user = null;

    if (connection!=null){
        String sql = "select * from smbms_user where userCode=?";
        Object[] params = {userCode};
        //System.out.println(userPassword);
        rs = BaseDao.execute(connection,sql,params,rs,pstmt);
        if (rs.next()){
```

通过 username 查用户的所有数据

```
if (connection!=null){
    String sql = "select * from smbms_user where userCode=?";    sql: "select * from smbms_user where userCode=?"
    Object[] params = {userCode};    userCode: "111"    params: Object[1]@4175
    //System.out.println(userPassword);
    rs = BaseDao.execute(connection,sql,params,rs,pstmt);    connection: ConnectionImpl@4174    sql: "select * from smbms_user wher
    if (rs.next()){
        user = new User();
        user.setId(rs.getInt( columnLabel: "id"));
        user.setUserCode(rs.getString( columnLabel: "userCode"));
        user.setUserName(rs.getString( columnLabel: "userName"));
        user.setUserPassword(rs.getString( columnLabel: "userPassword"));
        user.setGender(rs.getInt( columnLabel: "gender"));
        user.setBirthday(rs.getDate( columnLabel: "birthday"));
        user.setPhone(rs.getString( columnLabel: "phone"));
        user.setAddress(rs.getString( columnLabel: "address"));
        user.setUserRole(rs.getInt( columnLabel: "userRole"));
        user.setCreatedBy(rs.getInt( columnLabel: "createdBy"));
        user.setCreationDate(rs.getTimestamp( columnLabel: "creationDate"));
        user.setModifyBy(rs.getInt( columnLabel: "modifyBy"));
        user.setModifyDate(rs.getTimestamp( columnLabel: "modifyDate"));    user: null
    }

    BaseDao.closeResource( connection: null,pstmt,rs);    pstmt: null    rs: "com.mysql.cj.jdbc.result.ResultSetImpl@15651bf2"
```

如果查不到，走这段逻辑

```
if (user!=null){ //查有此人，可以登录
    //将用户的信息放到Session中；
    req.getSession().setAttribute(Constants.USER_SESSION,user);
    //跳转到主页重定向
    resp.sendRedirect( location: "jsp/frame.jsp");
}else {//查无此人，无法登录
    //转发回登录页面，顺带提示它，用户名或者密码错误；
    req.setAttribute( name: "error", o: "用户名或者密码不正确，憋批！");
    req.getRequestDispatcher( path: "login.jsp").forward(req,resp);
}
```

用户名或者密码不正确，憋批！

用户名:

密码:

如果能查到

```
rs = BaseDao.execute(connection,sql,params,rs,pstm); connection: ConnectionImpl@4286 pstm: null sql: "select * fr
if (rs.next()){
    user = new User();
    user.setId(rs.getInt( columnLabel: "id"));
    user.setUserCode(rs.getString( columnLabel: "userCode"));
    user.setUserName(rs.getString( columnLabel: "userName"));
    user.setUserPassword(rs.getString( columnLabel: "userPassword")); rs: "com.mysql.cj.jdbc.result.ResultSetImpl@1130e4
    user.setGender(rs.getInt( columnLabel: "gender"));
    user.setBirthday(rs.getDate( columnLabel: "birthday"));
    user.setPhone(rs.getString( columnLabel: "phone"));
    user.setAddress(rs.getString( columnLabel: "address"));
    user.setUserRole(rs.getInt( columnLabel: "userRole"));
    user.setCreatedBy(rs.getInt( columnLabel: "createdBy"));
    user.setCreationDate(rs.getTimestamp( columnLabel: "creationDate"));
    user.setModifyBy(rs.getInt( columnLabel: "modifyBy"));
    user.setModifyDate(rs.getTimestamp( columnLabel: "modifyDate"));
}
BaseDao.closeResource( connection: null,pstm,rs);
```

一个个设置实体类的参数

最后把这个实体类 return 回去

```
    user.setCreatedBy(rs.getInt( columnLabel: "createdBy"));
    user.setCreationDate(rs.getTimestamp( columnLabel: "creati
    user.setModifyBy(rs.getInt( columnLabel: "modifyBy"));
    user.setModifyDate(rs.getTimestamp( columnLabel: "modifyDa
}
BaseDao.closeResource( connection: null,pstm,rs);
return user;
```


然后调用实体类的基础 get 方法获取 password 进行比较

```
}  
//匹配密码  
if(null != user){  
    if(!user.getUserPassword().equals(password)) password: "123" user: User@4212  
        user = null;  
}  
return user;
```

最后又跳回 controller 层

走了登录成功的逻辑

```
userService userService = new UserServiceImpl(); userService: UserServiceImpl@4204  
User user = userService.login(userCode, userPassword); //这里已经把登录的人给查出来了 userService: UserServiceImpl@4204  
System.out.println(userCode); userCode: "wen"  
System.out.println(userPassword); userPassword: "123"  
if (user!=null){ //查有此人，可以登录  
    //将用户的信息放到Session中;  
    req.getSession().setAttribute(Constants.USER_SESSION,user); req: RequestFacade@4154 user: User@4212  
    //跳转到主页重定向  
    resp.sendRedirect( location: "jsp/frame.jsp"); resp: ResponseFacade@4155
```



这里是我自己操作的，我还是跟着他的流程走一遍

关键是学习开发思路

有了开发思路之后对白盒审计很有好处

然后我这里算了一下他的流程

他还剩下的课时是九个多小时

因为这个项目是 javaweb 的基础 也是后面学框架的基础

所以必须全部看完

我这里开两倍速 算了一下时间

全部看完视频还需要 4.38 个小时

现在是凌晨 2:38

也就是早上七点多能看完

留出 coding 时间

算八点吧

学的时候建议这样估算一下 有利于流程把控

Basedao 放基础的方法

Open

Select

Update

Close

再写个过滤器

Filer 把编码问题解决掉

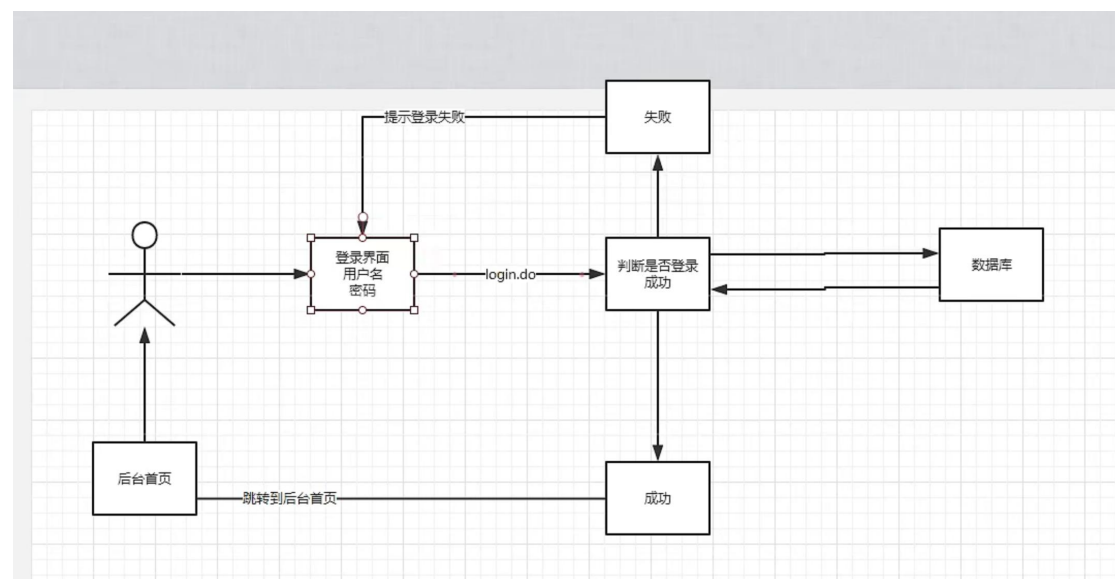
```
package com.sxt.servlet;  
  
import ...  
  
public class CharacterEncodingFilter implements Filter {  
  
    public void init(FilterConfig filterConfig) throws ServletException {  
    }  
  
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  
        request.setCharacterEncoding("utf-8");  
        response.setCharacterEncoding("utf-8");  
        chain.doFilter(request, response);  
    }  
  
    public void destroy() {  
    }  
}
```

再导入静态资源

直接导入即可

不赘述

登录模块



和刚刚我自己推的一样

先写 dao 层//后台查用户的数据

再写业务层//controller 层

Dao 层先写接口约束

再写 impl 实现

业务层先调 dao 层

```

public User login(String userCode, String password) {
    Connection connection = null;
    User user = null;

    try {
        connection = BaseDao.getConnection();
        //通过业务层调用对应的具体的数据库操作
        user = userDao.getLoginUser(connection, userCode);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        BaseDao.closeResource(connection, preparedStatement: null, resultSet: null);
    }
}

```

然后 return 一个 user

```

        BaseDao.closeResource(connection, preparedStatement: null, resultSet: null);
    }
    //匹配密码
    if(null != user){
        if(!user.getUserPassword().equals(password))
            user = null;
    }
    return user;
}

```

整个看他写下来

先写 dao 层：实现用户接口以及查询用户的 impl 类

再写 service 层：实现查询用户的逻辑，调用 dao 层

再写 controller 层的 servlet：实现用户 password 判断，把前端传参和后端查询的 password 比对

再写 view 层的 jsp：给用户一个传参数的地方

大体是这么个逻辑

登录优化

注销&权限过滤

注销

```

public class LogoutServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        //移除用户的session
        req.getSession().removeAttribute(Constants.USER_SESSION);
        resp.sendRedirect(location: req.getContextPath()+"/Login.jsp");//返回登录页面
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        doGet(req, resp);
    }
}

```

就是删除服务端中的 session

还有个过滤器，判断用户是否登录，禁止直接访问后台

```
public void init(FilterConfig filterConfig) throws ServletException {  
    }  
  
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws  
        HttpServletRequest request = (HttpServletRequest) req;  
        HttpServletResponse response = (HttpServletResponse) resp;  
  
        //过滤器，从Session中获取用户，  
        User user = (User) request.getSession().getAttribute(Constants.USER_SESSION);  
  
        if (user==null){ //已经被移除或者注销了，或者未登录  
            response.sendRedirect( location: "/smbms/error.jsp");  
        }else {  
            chain.doFilter(req,resp);  
        }  
    }  
}
```

设置 web.xml 路由

```
<filter>  
    <filter-name>Sysfilter</filter-name>  
    <filter-class>com.mario.filter.SysFilter</filter-class>  
</filter>  
<filter-mapping>  
    <filter-name>Sysfilter</filter-name>  
    <url-pattern>/jsp/*</url-pattern>  
</filter-mapping>
```

密码修改

先写 dao

```
public int updatePwd(Connection connection, int id, String userPassword) throws SQLException {  
    int updateRows=0;  
    PreparedStatement pstmt = null;  
    if(connection!=null){  
        String Sql="UPDATE `smbms_user` SET `userPassword`=? WHERE `id`=? ";  
        Object []params={userPassword,id};  
        updateRows=BaseDao.execute(connection,Sql,params,pstmt);  
    }  
    BaseDao.closeResource( connection: null,pstmt, resultSet: null);  
    return updateRows;  
}
```

Service 调用

```

public boolean updatePwd(int id, String password) {
    boolean flag = false;
    Connection connection = null;
    try{
        connection = BaseDao.getConnection();
        if(UserDao.updatePwd(connection,id,password) > 0)
            flag = true;
    }catch (Exception e) {
        e.printStackTrace();
    }finally{
        BaseDao.closeResource(connection, preparedStatement: null, resultSet: null);
    }
    return flag;
}

```

这里定义了一个 flag

如果修改成功

Flag 就为 true

然后作者在这里排错排了半天

哈哈哈

```

String newpassword = req.getParameter( name: "newpassword");
boolean flag= false;
//判断是否有这个用户是否存在，以及新密码不为空
if(attribute!=null && !StringUtils.isEmpty(newpassword)){
    UserService userService = new UserServiceImpl();
    flag = userService.updatePwd(((User) attribute).getId(), newpassword);
    if (flag) {
        req.setAttribute(Constants.SYS_MESSAGE, o: "修改密码成功，请退出后重新登录");
        //密码修改成功后移除当前session
        req.getSession().removeAttribute(Constants.USER_SESSION);
    }else{
        req.setAttribute(Constants.SYS_MESSAGE, o: "密码修改失败请重新输入");
    }
}
}else{
    req.setAttribute(Constants.SYS_MESSAGE, o: "新密码设置错误请重新输入");
}

```

小伙子这里少了一个!

真的服气

哈哈哈

找了半小时

然后根据 flag 判断是否修改成功

优化密码注册

判断旧密码


```
$.ajax({
    type: "GET",
    url: path + "/jsp/user.do",
    data: {method: "pwdmodify", oldpassword: oldpassword.val()}, //data就是ajax传递的参数
    /*
    上面这句话等价于 path + "/jsp/user.do?method=pwdmodify&&oldpassword=oldpassword.val()"
    */
    dataType: "json", //主调开发都是用JSON实现前后端开发{}
    success: function(data){
        if(data.result == "true"){//旧密码正确
            validateTip(oldpassword.next(), css: {"color": "green"}, imgYes, status: true);
        } else if(data.result == "false"){//旧密码输入不正确
            validateTip(oldpassword.next(), css: {"color": "red"}, tipString: imgNo + " 原密码输入不正确", status: false);
        } else if(data.result == "sessionerror"){//当前用户session过期，请重新登录
            validateTip(oldpassword.next(), css: {"color": "red"}, tipString: imgNo + " 当前用户session过期，请重新登录", status: false);
        } else if(data.result == "error"){//旧密码输入为空
            validateTip(oldpassword.next(), css: {"color": "red"}, tipString: imgNo + " 请输入旧密码", status: false);
        }
    }
},
```

这里作者用了一套 ajax 来传参实现验证老密码

这个 json 传参有点意思

果不其然 pom 文件里就有好东西

```
<!-- 阿里fastjson依赖-->
<!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.62</version>
</dependency>
</dependencies>
```

哈哈

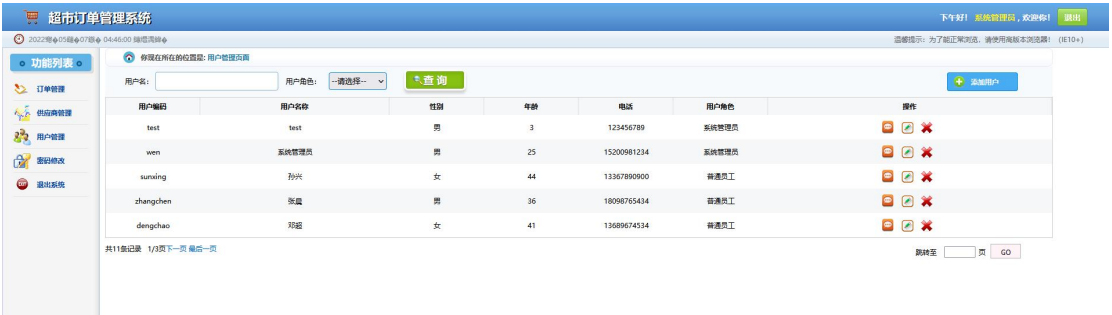
他这里校验完毕

```
private void modifyPwd(HttpServletRequest req, HttpServletResponse resp){
    //从session中获得用户的旧密码,这里的attribute包括了用户的所用信息
    Object attribute = req.getSession().getAttribute(Constants.USER_SESSION);
    //从前端输入的页面中获得输入的旧密码
    String oldpassword = req.getParameter( name: "oldpassword");
    //万能的Map
    Map<String, String> resultMap = new HashMap<>();
    if (attribute==null){//取到的session为空,意味着session过期了
        resultMap.put("result", "sessionerror");
    } else if (StringUtils.isEmpty(oldpassword)){//如果输入的旧密码为空
        resultMap.put("result", "sessionerror");
    } else{//session不为空,输入的旧密码也不为空,则取出当前旧密码与之比较
        String userPassword = ((User) attribute).getUserPassword();
        if(oldpassword.equals(userPassword)){
            resultMap.put("result", "true");
        } else {
            resultMap.put("result", "false");
        }
    }
}
```

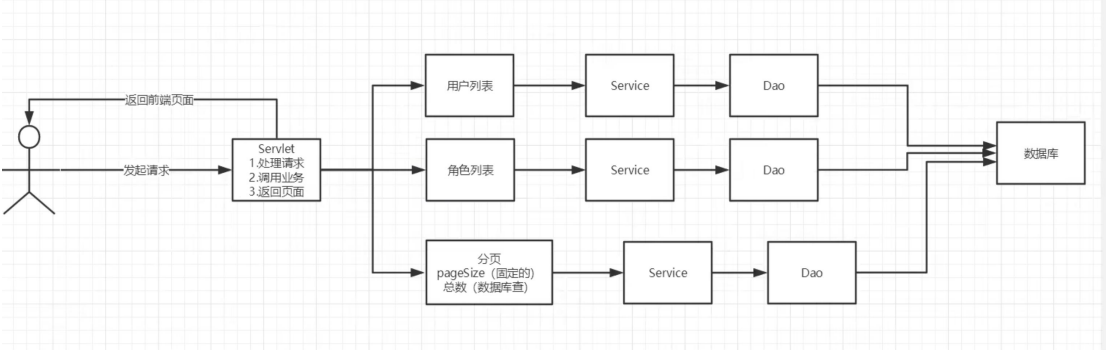
然后返回状态码给前端，前端再根据状态码显示页面
最后进行 json 转换，用 json 格式 write 过去

```
try {
    resp.setContentType("application/json");
    PrintWriter out = resp.getWriter();
    out.write(JSONArray.toJSONString(resultMap));
    out.flush();
    out.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

用户管理底层实现



架构



先导入分页工具类
然后导入用户前端 view 的 jsp
然后再来写 dao 层

```
// 1. 导入分页工具类
public int getUserCount(Connection connection, String userName, int userRole) throws Exception {
    int count=0;
    PreparedStatement pstmt = null;
    ResultSet rs=null;
    if (connection!=null) {
        StringBuffer sql=new StringBuffer();
        sql.append("SELECT COUNT(*) AS count FROM `smbms_user` u, `smbms_role` r WHERE u.`userRole`=r.`id`");
        ArrayList<Object> list = new ArrayList<>(); //存放可能会放进sql里的参数，就是用未替代?的params
        if(!StringUtil.isEmpty(userName)){
            sql.append(" and u.username like ?");
            list.add("%"+userName+"%");//模糊查询，index:0
        }
        if(userRole>0){
            sql.append(" and u.userRole = ?");
            list.add(userRole);//index:1
        }
    }
}
```


他这个 dao 层 就有 sql 注入

Id 这里传参过去即可

后面这里跟了两段 sql

```
ArrayList<Object> list = new ArrayList<>(); // 存放可能
if(!StringUtil.isNullOrEmpty(userName)){
    sql.append(" and u.username like ?");
    list.add("%"+userName+"%");//模糊查询, index:0
}
if(userRole>0){
    sql.append(" and u.userRole = ?");
    list.add(userRole);//index:1
}
```

Append 过去的

最后执行

```
Object[] params = list.toArray();//转换成数组
System.out.println("当前的sql语句为----->"+sql);
rs = BaseDao.execute(connection, sql.toString(), params, rs, pstmt);
if(rs.next()){
    count=rs.getInt( columnLabel: "count");
}
BaseDao.closeResource( connection: null, pstmt, rs);
```

用户角色列表的建立

```
public class RoleDaoImpl implements RoleDao {
    public List<Role> getRoleList(Connection connection) throws Exception {
        PreparedStatement pstmt=null;
        ResultSet rs=null;
        List<Role> roleList = new ArrayList<>();
        if(connection!=null){
            String sql="SELECT * FROM `smbms_role`";
            Object[] params={};
            rs = BaseDao.execute(connection, sql, params, rs, pstmt);
            while(rs.next()){
                Role role = new Role();
                role.setId(rs.getInt( columnLabel: "id"));
                role.setRoleCode(rs.getString( columnLabel: "roleCode"));
                role.setRoleName(rs.getString( columnLabel: "roleName"));
                roleList.add(role);
            }
        }
        BaseDao.closeResource( connection: null, pstmt, rs);
        return roleList;
        //细心细心细心
    }
}
```

这里还要单独写一个 pojo 实体类，方便操作

```

import java.util.Date;
public class Role {
    private Integer id; //id
    private String roleCode; //角色编码
    private String roleName; //角色名称
    private Integer createdBy; //创建者
    private Date creationDate; //创建时间
    private Integer modifyBy; //更新者
    private Date modifyDate; //更新时间

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getRoleCode() { return roleCode; }
    public void setRoleCode(String roleCode) { this.roleCode = roleCode; }
    public String getRoleName() { return roleName; }
    public void setRoleName(String roleName) { this.roleName = roleName; }
    public Integer getCreatedBy() { return createdBy; }
    public void setCreatedBy(Integer createdBy) { this.createdBy = createdBy; }
    public Date getCreationDate() { return creationDate; }
    public void setCreationDate(Date creationDate) { this.creationDate = creationDate; }
    public Integer getModifyBy() { return modifyBy; }
    public void setModifyBy(Integer modifyBy) { this.modifyBy = modifyBy; }
    public Date getModifyDate() { return modifyDate; }
    public void setModifyDate(Date modifyDate) { this.modifyDate = modifyDate; }
}

```

前端基本的增删改查和后端对应

增

```

//增加用户
private void add(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("当前正在执行增加用户操作");
    //从前端得到页面的请求的参数即用户输入的值
    String userCode = req.getParameter( name: "userCode");
    String userName = req.getParameter( name: "userName");
    String userPassword = req.getParameter( name: "userPassword");
    //String ruserPassword = req.getParameter("ruserPassword");
    String gender = req.getParameter( name: "gender");
    String birthday = req.getParameter( name: "birthday");
    String phone = req.getParameter( name: "phone");
    String address = req.getParameter( name: "address");
    String userRole = req.getParameter( name: "userRole");
    //把这些值塞进一个用户属性中
}

```

删

```

//删除用户，需要当前的Id，来找到这个用户然后删除
private void delUser(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException{
    String id = req.getParameter( name: "uid");
    Integer delId = 0;
    try{
        delId = Integer.parseInt(id);
    }catch (Exception e) {
        // TODO: handle exception
        delId = 0;
    }
    //需要判断是否能删除成功
    HashMap<String, String> resultMap = new HashMap<>();
    if(delId <= 0){
        resultMap.put("delResult", "notexist");
    }else{
        UserService userService = new UserServiceImpl();
        if(userService.deleteUserById(delId)){

```

改

```

//修改用户信息
private void modify(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException{
    //需要拿到前端传递进来的参数
    String id = req.getParameter( name: "uid");
    String userName = req.getParameter( name: "userName");
    String gender = req.getParameter( name: "gender");
    String birthday = req.getParameter( name: "birthday");
    String phone = req.getParameter( name: "phone");
    String address = req.getParameter( name: "address");
    String userRole = req.getParameter( name: "userRole");

    //创建一个user对象接收这些参数
    User user = new User();
    user.setId(Integer.valueOf(id));
    user.setUserName(userName);
    user.setGender(Integer.valueOf(gender));
    try {

```

查

```

//通过id得到用户信息
private void getUserById(HttpServletRequest req, HttpServletResponse resp,String url) throws ServletException, IOException{
    String id = req.getParameter( name: "uid");
    if(!StringUtils.isEmpty(id)){
        //调用后台方法得到user对象
        UserService userService = new UserServiceImpl();
        User user = userService.getUserById(id);
        req.setAttribute( name: "user", user);
        req.getRequestDispatcher(url).forward(req, resp);
    }
}

```

基本都是一套逻辑

前端传参

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String method = request.getParameter( name: "method");
    System.out.println("method----> " + method);
    if(method != null && method.equals("add")){
        //增加操作
        this.add(request, response);
    }else if(method != null && method.equals("query")){
        //查询用户操作
        this.query(request, response);
    }else if(method != null && method.equals("getrolelist")){
        //查询用户角色表
        this.getRoleList(request, response);
    }else if(method != null && method.equals("ucexist")){
        //查询当前用户编码是否存在
        this.userCodeExist(request, response);
    }else if(method != null && method.equals("deluser")){
        //删除用户
        this.delUser(request, response);
    }else if(method != null && method.equals("view")){
        //查看用户信息
    }
}
```

这里根据前端传参判断具体要执行什么操作

然后调用 this.xxx 方法

然后往 service 层走

```
user.setUserRole(Integer.valueOf(userRole));
user.setCreationDate(new Date());
//查找当前正在登陆的用户的id
user.setCreatedBy(((User)req.getSession().getAttribute(Constants.USER_SESSION)).getId());
UserService userService = new UserServiceImpl();
Boolean flag = userService.add(user);
//如果添加成功，则页面转发，否则重新刷新，再次跳转到当前页面
if(flag){
    resp.sendRedirect( location: req.getContextPath()+"/jsp/user.do?method=query");
}else{
    req.getRequestDispatcher( path: "useradd.jsp").forward(req,resp);
}
```

```
public Boolean add(User user) {
    boolean flag = false;
    Connection connection = null;
    try {
        connection = BaseDao.getConnection();//获得连接
        connection.setAutoCommit(false);//开启JDBC事务管理
        int updateRows = userDao.add(connection,user);
        connection.commit();
        if(updateRows > 0){
            flag = true;
            System.out.println("add success!");
        }else{
            System.out.println("add failed!");
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
    }
}
```


走到 service 层之，很明显看到，调用了 dao 层

```
//增加用户信息
public int add(Connection connection, User user) throws Exception;

//通过userId删除用户信息
public int deleteUserById(Connection connection, Integer delId) throws Exception;

//通过userId查看当前用户信息
public User getUserById(Connection connection, String id) throws Exception;
//修改用户信息
public int modify(Connection connection, User user) throws Exception;
```

然后然后到 impl 实现方法

```
public int add(Connection connection, User user) throws Exception {
    PreparedStatement pstmt=null;
    int updateNum=0;
    if(connection!=null) {
        String sql = "insert into smbms_user (userCode,userName,userPassword," +
            "userRole,gender,birthday,phone,address,creationDate,createdBy) " +
            "values(?,?,?,?,?,?,?,?,?,?)";
        Object[] params = {user.getUserCode(), user.getUserName(), user.getUserPassword(),
            user.getUserRole(), user.getGender(), user.getBirthday(),
            user.getPhone(), user.getAddress(), user.getCreationDate(), user.getCreatedBy()};
        updateNum = BaseDao.execute(connection, sql, params, pstmt);
        BaseDao.closeResource(connection: null, pstmt, resultSet: null);
    }
    return updateNum;
}
```

基本就是这套东西一直复用

一个路子

文件传输原理

基本 IO 操作

Input

Output

然后就是基本的上传

很简单

不赘述了

---20220507 7: 45---

Javaweb 基础这块都过完一遍了

中途还打了三把游戏 出去吃了个早饭

因为有些东西本来就会 所以就跳过了 节约了一些时间

先睡一觉 醒了之后会继续跟上框架的学习过程

