湖南石油化工職業技術學院
HUNAN PETROCHEMICAL VOCATIONAL TECHNOLOGY COLLEGE

《JSP/Servlet 动态网页》实验指导书

院系名称：经管信息学院

课程代码：31090820

总学时数：64

适用专业：大数据技术与应用专业

编 制 人：杨超

编制日期：2020 年 8 月

审 核 人：

审 定 人：

目　　录

一、 课程性质和教学目的

本课程是软件技术专业的专业基础课程,是培养学生 Web 后端动态页面设计能力的支撑课程。本课程主要培养学生的静态页面设计能力，该课程综合 Java 语言、JSP/Servlet 等技术，通过“教、学、做”理论与实践一体化教学，使学生掌握 Web 后端动态页面编写的基本方法，并逐步形成正确的后端程序设计思想，能够熟练地使用 Java 开发语言、JSP/Servlet 开发组件，为 Web 后端开发后续课程打下基础。

通过本课程的学习,使学生逐步建立和掌握 Web 后端动态页面设计的思想方法，具有分析问题和解决问题的能力，能够使用 Java 开发语言、JSP/Servlet 开发组件编写 Web 后端动态页面解决实际问题,具备吃苦耐劳、团结协作的良好品质。

二、实验目的

上机实验的目的不仅是为了验证教材和讲课的内容,或者验证自己所编写的程序正确与否。学习程序设计上机实验的目的是:

1．加深对讲授内容的理解，尤其是一些语法规定，课堂讲授即枯燥无味又难以记忆，但它们都很重要。能过多次上机就能自然地、熟练地掌握。通过上机掌握语法是行之有效的方法。

2．学会上机调试程序。即善于发现程序中的错误，并且能很快排除这些错误， 使程序能正确运行。要真正掌握这门课程，不仅应当了解和熟悉有关理论和方法， 还要求自己动手实现即会编程并上机调试通过。故应给予充分重视。调试程序固然可以借鉴他人的现成经验,但更重要的是通过自己的直接实践来累积经验， 而且有些经验是只能意会难以言传。调试程序的能力是每个程序设计人员应当掌握的一项基本功。

3．做实验时不要在程序通过后就认为搞定、完成任务了，而应当在已通过的程序上作一些改动（例如修改一些参数、增加程序一些功能、改变输入数据的方法等），以观察和分析所出现的情况。

三、上机实验前的准备工作

实验前应做好准备工作，以充分利用有限的上机时间。准备工作至少包括:

1．复习和掌握本实验有关的教学内容。

2．准备好上机所需的程序。初学者切忌不编写程序或抄别人的程序去上机，应从一开始就养成严谨的科学作风。

3．对运行上可能出现的问题应事先做出估计；对程序中自己有疑问的地方，应作上记号，以便在上机时给予注意。

4．根据实验内容认真准备实验程序及调试时所需的输入数据。

5．在上实验课之前必须写好预习报告（编程题源程序用纸写好或画好程序流程图）

6．填空与改错题等题要预先做好，上机时的工作只能是输入源程序和调试修改。

7.认真完成实验内容， 得出正确的实验结果。 实验结束后总结实验内容、书写实验报告。

8.遵守实验室规章制度、不缺席、按时上、下机。

四、实验环境

代码编写环境：可根据机房主机条件自己决定。推荐 notepad++、 IDEA 或 eclipse 等。

页面预览环境：建议学生及早考虑和适应跨浏览器自适应问题。推荐 Firefox、Google chrome、Internet Explorer 等。

实验 1 开发环境搭建

实验目的

1. 安装基本开发环境
2. 开发并运行简单的 Web 项目

实验内容

JDK 安装

如果是 exe 文件，双击安装

如果是 Zip 文件，解压到自定义路径，配置环境环境变量

环境变量配置：

我的电脑——》属性——》高级系统设置——》环境变量

1.1 在系统变量点击新建 JAVA_HOME 填入你的 JDK 安装路径

1.2 在系统变量中新建 CLASSPATH 填入.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar

1.3 在 path 中加入值，注意这里跟 win7 不同 不可以使用%JAVA_HOME%等符号引用地址，一定要使用绝对路径，这时将你的 jdk 的 bin 目录的路径和 jre 的 bin 目录的路径加在前面即可，注意用分号隔开

1.4 点击两次确定，保存完成

1.5 cmd 窗口运行：java -version

Tomcat

web 相关概念回顾

1. 软件架构
    1. C/S：客户端/服务器端
    2. B/S：浏览器/服务器端
2. 资源分类
    1. 静态资源：所有用户访问后，得到的结果都是一样的，称为静态资源.静态资源可以直接被浏览器解析
        * 如： html,css,JavaScript
    2. 动态资源:每个用户访问相同资源后，得到的结果可能不一样。

称为动态资源。动态资源被访问后，需要先转换为静态资源，在返回给浏览器

* 如：servlet/jsp,php,asp....

3. 网络通信三要素
    1. IP：电子设备(计算机)在网络中的唯一标识。
    2. 端口：应用程序在计算机中的唯一标识。 0~65536
    3. 传输协议：规定了数据传输的规则
        1. 基础协议：
            1. tcp:安全协议，三次握手。 速度稍慢
            2. udp：不安全协议。 速度快

web 服务器软件：

* 服务器：安装了服务器软件的计算机
* 服务器软件：接收用户的请求，处理请求，做出响应
* web 服务器软件：接收用户的请求，处理请求，做出响应。
* 在 web 服务器软件中，可以部署 web 项目，让用户通过浏览器来访问这些项目
* web 容器

* 常见的 java 相关的 web 服务器软件：
* webLogic:oracle 公司,大型的 JavaEE 服务器,支持所有的 JavaEE 规范，收费的。
* webSphere：IBM 公司，大型的 JavaEE 服务器，支持所有的 JavaEE 规范，收费的。
* JBOSS：JBOSS 公司的，大型的 JavaEE 服务器，支持所有的 JavaEE 规范，收费的。

* Tomcat：Apache 基金组织，中小型的 JavaEE 服务器，仅仅支持少量的 JavaEE 规范 servlet/jsp。开源的，免费的。

* JavaEE：Java 语言在企业级开发中使用的技术规范的总和，一共规定了 13 项大的规范

* Tomcat：web 服务器软件

1. 下载：http://tomcat.apache.org/
2. 安装：解压压缩包即可。
   * 注意：安装目录建议不要有中文和空格
3. 卸载：删除目录就行了
4. 启动：
   * bin/startup.bat ,双击运行该文件即可
   * 访问：浏览器输入：http://localhost:8080 回车访问自己
     http://别人的 ip:8080 访问别人

   * 可能遇到的问题：
     1. 黑窗口一闪而过：
        * 原因： 没有正确配置 JAVA_HOME 环境变量
        * 解决方案：正确配置 JAVA_HOME 环境变量

     2. 启动报错：
        1. 暴力：找到占用的端口号，并且找到对应的进程，杀死该进程
           * netstat -ano
        2. 温柔：修改自身的端口号
           * conf/server.xml
           * <Connector port="8888" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8445" />

* 一般会将 tomcat 的默认端口号修改为 80。80 端口号是 http 协议的默认端口号。

* 好处：在访问时，就不用输入端口号

5. 关闭：

1. 正常关闭：

* bin/shutdown.bat

* ctrl+c

2. 强制关闭：

* 点击启动窗口的×

6. 配置：

* 部署项目的方式：

1. 直接将项目放到 webapps 目录下即可。

* /hello：项目的访问路径-->虚拟目录

* 简化部署：将项目打成一个 war 包，再将 war 包放置到 webapps 目录下。

* war 包会自动解压缩

2. 配置 conf/server.xml 文件

在<Host>标签体中配置

<Context docBase="D:\hello" path="/hehe" />

* docBase:项目存放的路径

* path：虚拟目录

3. 在 conf\Catalina\localhost 创建任意名称的 xml 文件。在文件中编写

<Context docBase="D:\hello" />

* 虚拟目录：xml 文件的名称

* 静态项目和动态项目：

* 目录结构

* java 动态项目的目录结构：

-- 项目的根目录

-- WEB-INF 目录：

-- web.xml：web 项目的核心配置文件

-- classes 目录：放置字节码文件的目录

-- lib 目录：放置依赖的 jar 包

* 将 Tomcat 集成到 IDEA 中，并且创建 JavaEE 的项目，部署项目。

IDEA 与 tomcat 的相关配置

IDEA 会为每一个 tomcat 部署的项目单独建立一份配置文件

* 查看控制台的 log：

Using CATALINA_BASE: "C:\Users\fqy\.IntelliJIdea2018.1\system\tomcat_itcast"

工作空间项目和 tomcat 部署的 web 项目

* tomcat 真正访问的是“tomcat 部署的 web 项目”，"tomcat 部署的 web 项目"对 应着"工作空间项目" 的 web 目录下的所有资源

* WEB-INF 目录下的资源不能被浏览器直接访问。

断点调试：使用"小虫子"启动 dubug 启动

实验 2 SERVLET 入门使用

实验目的

1. 掌握 Servlet 的定义

2. 熟悉 Servlet 的使用

实验内容

* 快速入门：

1. 创建 JavaEE 项目
2. 定义一个类，实现 Servlet 接口
   * public class ServletDemo1 implements Servlet
3. 实现接口中的抽象方法
4. 配置 Servlet

在 web.xml 中配置：

```
<!--配置 Servlet -->
<servlet>
    <servlet-name>demo1</servlet-name>
<servlet-class>cn.itcast.web.servlet.ServletDemo1</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>demo1</servlet-name>
    <url-pattern>/demo1</url-pattern>
</servlet-mapping>
```

* 执行原理：

1. 当服务器接受到客户端浏览器的请求后，会解析请求 URL 路径，获取访问的 Servlet 的资源路径
2. 查找 web.xml 文件，是否有对应的<url-pattern>标签体内容。
3. 如果有，则在找到对应的<servlet-class>全类名
4. tomcat 会将字节码文件加载进内存，并且创建其对象
5. 调用其方法

* Servlet 中的生命周期方法：

1. 被创建：执行 init 方法，只执行一次

* Servlet 什么时候被创建？

* 默认情况下，第一次被访问时，Servlet 被创建

* 可以配置执行 Servlet 的创建时机。

* 在<servlet>标签下配置

1. 第一次被访问时，创建

* <load-on-startup>的值为负数

2. 在服务器启动时，创建

* <load-on-startup>的值为 0 或正整数

* Servlet 的 init 方法，只执行一次，说明一个 Servlet 在内存中只存在一个对象，Servlet 是单例的

* 多个用户同时访问时，可能存在线程安全问题。

* 解决：尽量不要在 Servlet 中定义成员变量。即使定义了成员变量，也不要对修改值

2. 提供服务：执行 service 方法，执行多次

* 每次访问 Servlet 时，Service 方法都会被调用一次。

3. 被销毁：执行 destroy 方法，只执行一次

* Servlet 被销毁时执行。服务器关闭时，Servlet 被销毁

* 只有服务器正常关闭时，才会执行 destroy 方法。

* destroy 方法在 Servlet 被销毁之前执行，一般用于释放资源

* Servlet3.0：

* 好处：

* 支持注解配置。可以不需要 web.xml 了。

* 步骤：

1. 创建 JavaEE 项目，选择 Servlet 的版本 3.0 以上，可以不创建 web.xml
2. 定义一个类，实现 Servlet 接口
3. 复写方法
4. 在类上使用@WebServlet 注解，进行配置

   * @WebServlet("资源路径")

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface WebServlet {
    String name() default "";//相当于<Servlet-name>

    String[] value() default {};//代表 urlPatterns()属性配置

    String[] urlPatterns() default {};//相当于<url-pattern>

    int loadOnStartup() default -1;//相当于<load-on-startup>

    WebInitParam[] initParams() default {};

    boolean asyncSupported() default false;

    String smallIcon() default "";
```

```
        String largeIcon() default "";

        String description() default "";

        String displayName() default "";
    }
```

* Servlet 的体系结构

Servlet -- 接口

|

GenericServlet -- 抽象类

|

HttpServlet -- 抽象类

* GenericServlet：将 Servlet 接口中其他的方法做了默认空实现，只将 service()方法作为抽象

* 将来定义 Servlet 类时，可以继承 GenericServlet，实现 service()方法即可

* HttpServlet：对 http 协议的一种封装，简化操作

1. 定义类继承 HttpServlet

2. 复写 doGet/doPost 方法

* Servlet 相关配置

1. urlpartten:Servlet 访问路径

1. 一个 Servlet 可以定义多个访问路径 ：@WebServlet({"/d4","/dd4","/ddd4"})

2. 路径定义规则：

1. /xxx：路径匹配

2. /xxx/xxx:多层路径，目录结构

3. *.do：扩展名匹配

实验 3 HTTP 协议及 REQUEST 对象的应用

实验目的

1. 熟悉 Http 协议的组帧
2. Request 的使用

实验内容

HTTP：

* 概念：Hyper Text Transfer Protocol 超文本传输协议
  * 传输协议：定义了，客户端和服务器端通信时，发送数据的格式
  * 特点：
    1. 基于 TCP/IP 的高级协议
    2. 默认端口号:80
    3. 基于请求/响应模型的:一次请求对应一次响应
    4. 无状态的：每次请求之间相互独立，不能交互数据

  * 历史版本：
    * 1.0：每一次请求响应都会建立新的连接
    * 1.1：复用连接

* 请求消息数据格式
  1. 请求行

     请求方式 请求 url 请求协议/版本

     GET /login.html HTTP/1.1

     * 请求方式：
       * HTTP 协议有 7 中请求方式，常用的有 2 种
         * GET：
           1. 请求参数在请求行中，在 url 后。
           2. 请求的 url 长度有限制的
           3. 不太安全

* POST：

1. 请求参数在请求体中
2. 请求的 url 长度没有限制的
3. 相对安全

2. 请求头：

客户端浏览器告诉服务器一些信息

请求头名称：请求头值

* 常见的请求头：

1. User-Agent：浏览器告诉服务器，我访问你使用的浏览器版本信息

* 可以在服务器端获取该头的信息，解决浏览器的兼容性问题

2. Referer：http://localhost/login.html

* 告诉服务器，我(当前请求)从哪里来？

* 作用：

1. 防盗链：
2. 统计工作：

3. 请求空行

空行，就是用于分割 POST 请求的请求头，和请求体的。

4. 请求体(正文)：

* 封装 POST 请求消息的请求参数的

* 字符串格式：

POST /login.html HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0

Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-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

Accept-Encoding: gzip, deflate

Referer: http://localhost/login.html

Connection: keep-alive

Upgrade-Insecure-Requests: 1

username=zhangsan

* 响应消息数据格式

Request：

1. request 对象和 response 对象的原理
    1. request 和 response 对象是由服务器创建的。我们来使用它们
    2. request 对象是来获取请求消息，response 对象是来设置响应消息

2. request 对象继承体系结构：

    ServletRequest -- 接口

    | 继承

    HttpServletRequest -- 接口

    | 实现

    org.apache.catalina.connector.RequestFacade 类(tomcat)

3. request 功能：
    1. 获取请求消息数据
        1. 获取请求行数据

* GET /day14/demo1?name=zhangsan HTTP/1.1

* 方法：

1. 获取请求方式 ：GET

* String getMethod()

2. (*)获取虚拟目录：/day14

* String getContextPath()

3. 获取 Servlet 路径: /demo1

* String getServletPath()

4. 获取 get 方式请求参数：name=zhangsan

* String getQueryString()

5. (*)获取请求 URI：/day14/demo1

* String getRequestURI(): /day14/demo1

* StringBuffer getRequestURL() :http://localhost/day14/demo1

* URL: 统一资源定位符 : http://localhost/day14/demo1 中华人民共和国

* URI：统一资源标识符 : /day14/demo1 共和国

6. 获取协议及版本：HTTP/1.1

* String getProtocol()

7. 获取客户机的 IP 地址：

* String getRemoteAddr()

2. 获取请求头数据

* 方法：

* (*)String getHeader(String name):通过请求头的名

称获取请求头的值

* Enumeration<String> getHeaderNames():获取所有的请求头名称

3. 获取请求体数据:

* 请求体：只有 POST 请求方式，才有请求体，在请求体中封装了 POST 请求的请求参数

* 步骤：

1. 获取流对象

* BufferedReader getReader():获取字符输入流，只能操作字符数据

* ServletInputStream getInputStream()：获取字节输入流，可以操作所有类型数据

* 在文件上传知识点后讲解

2. 再从流对象中拿数据

2. 其他功能：

1. 获取请求参数通用方式：不论 get 还是 post 请求方式都可以使用下列方法来获取请求参数

1. String getParameter(String name):根据参数名称获取参数值 username=zs&password=123

2. String[] getParameterValues(String name):根据参数名称获取参数值的数组 hobby=xx&hobby=game

3. Enumeration<String> getParameterNames():获取所有请求的参数名称

4. Map<String,String[]> getParameterMap():获取所有参数的 map 集合

* 中文乱码问题：

* get 方式：tomcat 8 已经将 get 方式乱码问题解决了

* post 方式：会乱码

* 解决：在获取参数前，设置 request 的编码 request.setCharacterEncoding("utf-8");

2. 请求转发：一种在服务器内部的资源跳转方式

1. 步骤：

1. 通过 request 对象获取请求转发器对象：RequestDispatcher getRequestDispatcher(String path)

2. 使用 RequestDispatcher 对象来进行转发：forward(ServletRequest request, ServletResponse response)

2. 特点：

1. 浏览器地址栏路径不发生变化

2. 只能转发到当前服务器内部资源中。

3. 转发是一次请求

3. 共享数据：

* 域对象：一个有作用范围的对象，可以在范围内共享数据

* request 域：代表一次请求的范围，一般用于请求转发的多个资源中共享数据

* 方法：

1. void setAttribute(String name,Object obj):存储数据

2. Object getAttitude(String name):通过键获取值

3. void removeAttribute(String name):通过键移除键值对

4. 获取 ServletContext：

* ServletContext getServletContext()

实验 4 REPONSE 的使用

实验目的

1．熟悉 Http 中 Response 的数据组包

2．熟悉 Response 的使用

实验内容

HTTP 协议：

1. 请求消息：客户端发送给服务器端的数据
    * 数据格式：
        1. 请求行
        2. 请求头
        3. 请求空行
        4. 请求体
2. 响应消息：服务器端发送给客户端的数据
    * 数据格式：
        1. 响应行
            1. 组成：协议/版本 响应状态码 状态码描述
            2. 响应状态码：服务器告诉客户端浏览器本次请求和响应的一个状态。
                1. 状态码都是 3 位数字
                2. 分类：
                    1. 1xx：服务器就收客户端消息，但没有接受完成，等待一段时间后，发送 1xx 多状态码
                    2. 2xx：成功。代表：200
                    3. 3xx：重定向。代表：302(重定向)，304(访问缓存)
                    4. 4xx：客户端错误。
                        * 代表：
                            * 404（请求路径没有对应的资源）
                            * 405：请求方式没有对应的 doXxx 方法

5. 5xx：服务器端错误。代表：500(服务器内部出现异常)

2. 响应头：

1. 格式：头名称： 值

2. 常见的响应头：

1. Content-Type：服务器告诉客户端本次响应体数据格式以及编码格式

2. Content-disposition：服务器告诉客户端以什么格式打开响应体数据

* 值：

* in-line:默认值,在当前页面内打开

* attachment;filename=xxx：以附件形式打开响应体。文件下载

3. 响应空行

4. 响应体:传输的数据

* 响应字符串格式

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Content-Length: 101
Date: Wed, 06 Jun 2018 07:08:42 GMT

<html>
  <head>
    <title>$Title$</title>
  </head>
```

<body>

hello , response

</body>

</html>

Response 对象

* 功能：设置响应消息

1. 设置响应行

1. 格式：HTTP/1.1 200 ok

2. 设置状态码：setStatus(int sc)

2. 设置响应头：

setHeader(String name, String value)

3. 设置响应体：

* 使用步骤：

1. 获取输出流

* 字符输出流：PrintWriter getWriter()

* 字节输出流：ServletOutputStream getOutputStream()

2. 使用输出流，将数据输出到客户端浏览器

* 案例：

1. 完成重定向

* 重定向：资源跳转的方式

* 代码实现：

```
//1. 设置状态码为 302
response.setStatus(302);
//2.设置响应头 location
```

response.setHeader("location","/day15/responseDemo2");

```
//简单的重定向方法
response.sendRedirect("/day15/responseDemo2");
```

* 重定向的特点:redirect
    1. 地址栏发生变化
    2. 重定向可以访问其他站点(服务器)的资源
    3. 重定向是两次请求。不能使用 request 对象来共享数据
* 转发的特点：forward
    1. 转发地址栏路径不变
    2. 转发只能访问当前服务器下的资源
    3. 转发是一次请求，可以使用 request 对象来共享数据

* forward 和 redirect 区别

* 路径写法：
    1. 路径分类
        1. 相对路径：通过相对路径不可以确定唯一资源
            * 如：./index.html
            * 不以/开头，以.开头路径

            * 规则：找到当前资源和目标资源之间的相对位置关系
                * ./：当前目录
                * ../:后退一级目录
        2. 绝对路径：通过绝对路径可以确定唯一资源
            * 如 ： http://localhost/day15/responseDemo2

/day15/responseDemo2

* 以/开头的路径

* 规则：判断定义的路径是给谁用的？判断请求将来从哪儿发出

* 给客户端浏览器使用：需要加虚拟目录(项目的访问路径)

* 建议虚拟目录动态获取：request.getContextPath()

* <a>，<form> 重定向...

* 给服务器使用：不需要加虚拟目录

* 转发路径

2. 服务器输出字符数据到浏览器

* 步骤：

1. 获取字符输出流

2. 输出数据

* 注意：

* 乱码问题：

1. PrintWriter pw = response.getWriter();获取的流的默认编码是ISO-8859-1

2. 设置该流的默认编码

3. 告诉浏览器响应体使用的编码

//简单的形式，设置编码，是在获取流之前设置

response.setContentType("text/html;charset=utf-8");

3. 服务器输出字节数据到浏览器

* 步骤：

1. 获取字节输出流
2. 输出数据

4. 验证码
    1. 本质：图片
    2. 目的：防止恶意表单注册

ServletContext 对象：

1.概念：

代表整个 web 应用，可以和程序的容器(服务器)来通信

2.获取：

1. 通过 request 对象获取

   request.getServletContext();

2. 通过 HttpServlet 获取

   this.getServletContext();

3.功能：

1. 获取 MIME 类型：

   * MIME 类型:在互联网通信过程中定义的一种文件数据类型

      * 格式： 大类型/小类型 text/html image/jpeg

   * 获取：String getMimeType(String file)

2. 域对象：共享数据

    1. setAttribute(String name,Object value)
    2. getAttribute(String name)
    3. removeAttribute(String name)

   * ServletContext 对象范围：所有用户所有请求的数据

3. 获取文件的真实(服务器)路径

    1. 方法：String getRealPath(String path)

```
String b = context.getRealPath("/b.txt");//web目录下资源访问
System.out.println(b);

String c = context.getRealPath("/WEB-INF/c.txt");//WEB-INF目录下的资源访问
System.out.println(c);

String a = context.getRealPath("/WEB-INF/classes/a.txt");//src目录下的资源访问
System.out.println(a);
```

案例：

* 文件下载需求：
    1. 页面显示超链接
    2. 点击超链接后弹出下载提示框
    3. 完成图片文件下载

* 分析：
    1. 超链接指向的资源如果能够被浏览器解析，则在浏览器中展示，如果不能解析，则弹出下载提示框。不满足需求
    2. 任何资源都必须弹出下载提示框
    3. 使用响应头设置资源的打开方式：
        * content-disposition:attachment;filename=xxx

* 步骤：
    1. 定义页面，编辑超链接href属性，指向Servlet，传递资源名称

filename

2. 定义 Servlet

1. 获取文件名称

2. 使用字节输入流加载文件进内存

3. 指定 response 的响应头：content-disposition:attachment;filename=xxx

4. 将数据写出到 response 输出流

* 问题：

* 中文文件问题

* 解决思路：

1. 获取客户端使用的浏览器版本信息

2. 根据不同的版本信息，设置 filename 的编码方式不同

实验 5 COOKIE 的应用

实验目的

1. 熟悉 Cookie 的原理

2. 熟悉 Cookie 的使用

实验内容

会话技术

1. 会话：一次会话中包含多次请求和响应。

* 一次会话：浏览器第一次给服务器资源发送请求，会话建立，直到有一方断开为止

2. 功能：在一次会话的范围内的多次请求间，共享数据

3. 方式：

1. 客户端会话技术：Cookie

2. 服务器端会话技术：Session

Cookie：

概念：

客户端会话技术，将数据保存到客户端

2. 快速入门：

* 使用步骤：

1. 创建 Cookie 对象，绑定数据

* new Cookie(String name, String value)

2. 发送 Cookie 对象

* response.addCookie(Cookie cookie)

3. 获取 Cookie，拿到数据

* Cookie[] request.getCookies()

3. 实现原理

* 基于响应头 set-cookie 和请求头 cookie 实现

4. cookie 的细节

1. 一次可不可以发送多个 cookie?

* 可以

* 可以创建多个 Cookie 对象，使用 response 调用多次 addCookie 方法发送 cookie 即可。

2. cookie 在浏览器中保存多长时间？

1. 默认情况下，当浏览器关闭后，Cookie 数据被销毁

2. 持久化存储：

* setMaxAge(int seconds)

1. 正数：将 Cookie 数据写到硬盘的文件中。持久化存储。并指定 cookie 存活时间，时间到后，cookie 文件自动失效

2. 负数：默认值

3. 零：删除 cookie 信息

3. cookie 能不能存中文？

* 在 tomcat 8 之前 cookie 中不能直接存储中文数据。

* 需要将中文数据转码---一般采用 URL 编码(%E3)

* 在 tomcat 8 之后，cookie 支持中文数据。特殊字符还是不支持，建议使用 URL 编码存储，URL 解码解析

4. cookie 共享问题？

1. 假设在一个 tomcat 服务器中，部署了多个 web 项目，那么在这些 web 项目中 cookie 能不能共享？

* 默认情况下 cookie 不能共享

* setPath(String path):设置 cookie 的获取范围。默认情况下，设置当前的虚拟目录

* 如果要共享，则可以将 path 设置为"/"

2. 不同的 tomcat 服务器间 cookie 共享问题？

* setDomain(String path):如果设置一级域名相同，那么多个服务器之间 cookie 可以共享

* setDomain(".baidu.com"),那么 tieba.baidu.com 和 news.baidu.com 中 cookie 可以共享

5. Cookie 的特点和作用

1. cookie 存储数据在客户端浏览器

2. 浏览器对于单个 cookie 的大小有限制(4kb) 以及 对同一个域名下的总 cookie 数量也有限制(20 个)

* 作用：

1. cookie 一般用于存出少量的不太敏感的数据

2. 在不登录的情况下，完成服务器对客户端的身份识别

6. 案例：记住上一次访问时间

1. 需求：

1. 访问一个Servlet，如果是第一次访问，则提示：您好，欢迎您首次访问。

2. 如果不是第一次访问，则提示：欢迎回来，您上次访问时间为:显示时间字符串

2. 分析：

1. 可以采用Cookie来完成

2. 在服务器中的Servlet判断是否有一个名为lastTime的cookie

1. 有：不是第一次访问

1. 响应数据：欢迎回来，您上次访问时间为:2018年6月10日11:50:20

2. 写回Cookie：lastTime=2018年6月10日11:50:01

2. 没有：是第一次访问

1. 响应数据：您好，欢迎您首次访问

2. 写回Cookie：lastTime=2018年6月10日11:50:01

3. 代码实现：

```
package cn.itcast.cookie;


import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.text.SimpleDateFormat;
import java.util.Date;


@WebServlet("/cookieTest")
public class CookieTest extends HttpServlet {
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        //设置响应的消息体的数据格式以及编码

response.setContentType("text/html;charset=utf-8");

        //1.获取所有 Cookie
        Cookie[] cookies = request.getCookies();
        boolean flag = false;//没有 cookie 为 lastTime
        //2.遍历 cookie 数组
        if(cookies != null && cookies.length > 0){
            for (Cookie cookie : cookies) {
                //3.获取 cookie 的名称
                String name = cookie.getName();
                //4.判断名称是否是：lastTime
                if("lastTime".equals(name)){
                    //有该 Cookie，不是第一次访问

                    flag = true;//有 lastTime 的 cookie

                    //设置 Cookie 的 value
```

```
                    //获取当前时间的字符串，重新设置Cookie
的值，重新发送cookie
                    Date date  = new Date();
                        SimpleDateFormat  sdf  =  new
SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
                    String str_date = sdf.format(date);
                        System.out.println("编码前：
"+str_date);
                    //URL编码
                                                str_date  =
URLEncoder.encode(str_date,"utf-8");
                        System.out.println("编码后：
"+str_date);
                    cookie.setValue(str_date);
                    //设置cookie的存活时间
                     cookie.setMaxAge(60 * 60 * 24 * 30);//
一个月
                    response.addCookie(cookie);



                    //响应数据
                    //获取Cookie的value，时间
                    String value = cookie.getValue();
                    System.out.println("解码前："+value);
                    //URL解码：
                                                  value  =
URLDecoder.decode(value,"utf-8");
                    System.out.println("解码后："+value);
                     response.getWriter().write("<h1>欢迎回
```

```
来，您上次访问时间为:"+value+"</h1>");

                        break;

                    }
                }
            }


            if(cookies == null || cookies.length == 0 || flag ==
false){
                //没有，第一次访问

                //设置 Cookie 的 value
                //获取当前时间的字符串，重新设置 Cookie 的值，重
新发送 cookie
                Date date  = new Date();
                                SimpleDateFormat  sdf  =  new
SimpleDateFormat("yyyy 年 MM 月 dd 日 HH:mm:ss");
                String str_date = sdf.format(date);
                System.out.println("编码前： "+str_date);
                //URL 编码
                str_date = URLEncoder.encode(str_date,"utf-8");
                System.out.println("编码后： "+str_date);

                Cookie cookie = new Cookie("lastTime",str_date);
                //设置 cookie 的存活时间
                cookie.setMaxAge(60 * 60 * 24 * 30);//一个月
                response.addCookie(cookie);
```

```
                response.getWriter().write("<h1>您好，欢迎您首次访问</h1>");
            }


        }

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

实验 6 SESSION 的应用

实验目的

1. 熟悉 Session 的原理
2. 熟悉 Session 的应用

实验内容

案例:改造 Cookie 案例

Session:

概念：

服务器端会话技术，在一次会话的多次请求间共享数据，将数据保存在服务器端的对象中。HttpSession

2. 快速入门：

1. 获取 HttpSession 对象：

```
HttpSession session = request.getSession();
```

2. 使用 HttpSession 对象：

```
Object getAttribute(String name)
```

void setAttribute(String name, Object value)

void removeAttribute(String name)

3. 原理

* Session 的实现是依赖于 Cookie 的。

4. 细节：

1. 当客户端关闭后，服务器不关闭，两次获取 session 是否为同一个？

* 默认情况下。不是。

* 如果需要相同，则可以创建 Cookie,键为 JSESSIONID，设置最大存活时间，让 cookie 持久化保存。

```
Cookie c = new Cookie("JSESSIONID",session.getId());
 c.setMaxAge(60*60);
 response.addCookie(c);
```

2. 客户端不关闭,服务器关闭后,两次获取的 session 是同一个吗？

* 不是同一个，但是要确保数据不丢失。tomcat 自动完成以下工作

* session 的钝化：

* 在服务器正常关闭之前，将 session 对象系列化到硬盘上

* session 的活化：

* 在服务器启动后，将 session 文件转化为内存中的 session 对象即可。

3. session 什么时候被销毁？

1. 服务器关闭

2. session 对象调用 invalidate() 。

3. session 默认失效时间 30 分钟

选择性配置修改

<session-config>

<session-timeout>30</session-timeout>

</session-config>

5. session 的特点

1. session 用于存储一次会话的多次请求的数据，存在服务器端
2. session 可以存储任意类型，任意大小的数据

* session 与 Cookie 的区别：

1. session 存储数据在服务器端，Cookie 在客户端
2. session 没有数据大小限制，Cookie 有
3. session 数据安全，Cookie 相对于不安全

案例：验证码

1. 案例需求：

1. 访问带有验证码的登录页面 login.jsp
2. 用户输入用户名，密码以及验证码。

* 如果用户名和密码输入有误，跳转登录页面，提示:用户名或密码错误

* 如果验证码输入有误，跳转登录页面，提示：验证码错误

* 如果全部输入正确，则跳转到主页 success.jsp，显示：用户名,欢迎您。

实验 7 JSP 页面开发

实验目的

1. 熟悉 JSP 指令和脚本

2. 熟悉 JSP 开发页面

实验内容

JSP:

1. 指令

* 作用:

用于配置 JSP 页面，导入资源文件

* 格式:

<%@ 指令名称 属性名 1=属性值 1 属性名 2=属性值 2 ... %>

* 分类:

1. page ： 配置 JSP 页面的

* contentType: 等同于 response.setContentType()

1. 设置响应体的 mime 类型以及字符集

2. 设置当前 jsp 页面的编码(只能是高级的 IDE 才能生效，如果使用低级工具，则需要设置 pageEncoding 属性设置当前页面的字符集)

* import: 导包

* errorPage: 当前页面发生异常后，会自动跳转到指定的错误页面

* isErrorPage: 标识当前也是是否是错误页面。

* true: 是，可以使用内置对象 exception

* false: 否。默认值。不可以使用内置对象 exception

2. include ： 页面包含的。导入页面的资源文件

* <%@include file="top.jsp"%>

3. taglib ： 导入资源

* <%@ taglib prefix="c"

uri="http://java.sun.com/jsp/jstl/core" %>

* prefix：前缀，自定义的

2. 注释：

1. html 注释：

<!-- -->:只能注释 html 代码片段

2. jsp 注释：推荐使用

<%-- --%>：可以注释所有

3. 内置对象

* 在 jsp 页面中不需要创建，直接使用的对象

* 一共有 9 个：

变量名	真实类型	作用
* pageContext	PageContext	当前页面共享数据，还可以获取其他八个内置对象
* request	HttpServletRequest	一次请求访问的多个资源(转发)
* session	HttpSession	一次会话的多个请求间
* application	ServletContext	所有用户间共享数据
* response	HttpServletResponse	响应对象
* page	Object	当前页面(Servlet)的对象 this
* out	JspWriter	输出对象，数据输出到页面上
* config	ServletConfig	

Servlet 的配置对象

* exception Throwable 异常对象

MVC：开发模式

1. jsp 演变历史

1. 早期只有 servlet，只能使用 response 输出标签数据，非常麻烦

2. 后来又 jsp，简化了 Servlet 的开发，如果过度使用 jsp，在 jsp 中即写大量的 java 代码，有写 html 表，造成难于维护，难于分工协作

3. 再后来，java 的 web 开发，借鉴 mvc 开发模式，使得程序的设计更加合理性

2. MVC：

1. M：Model，模型。JavaBean

* 完成具体的业务操作，如：查询数据库，封装对象

2. V：View，视图。JSP

* 展示数据

3. C：Controller，控制器。Servlet

* 获取用户的输入

* 调用模型

* 将数据交给视图进行展示

* 优缺点：

1. 优点：

1. 耦合性低，方便维护，可以利于分工协作

2. 重用性高

2. 缺点：

1. 使得项目架构变得复杂，对开发人员要求高。

实验 8 案例开发：信息列表显示

实验目的

1. 熟悉案例开发的技术需求
2. 熟悉案例开发的流程

实验内容

三层架构：软件设计架构

1. 界面层(表示层)：用户看的得界面。用户可以通过界面上的组件和服务器进行交互
2. 业务逻辑层：处理业务逻辑的。
3. 数据访问层：操作数据存储文件。

案例：用户信息列表展示

1. 需求：用户信息的增删改查操作
2. 设计：
    1. 技术选型：Servlet+JSP+MySQL+JDBCTempleat+Duird+BeanUtilS+tomcat
    2. 数据库设计：

```
create database javaweb; -- 创建数据库
use javaweb;             -- 使用数据库
create table user(   -- 创建表
    id int primary key auto_increment,
    name varchar(20) not null,
    gender varchar(5),
    age int,
    address varchar(32),
    qq  varchar(20),
    email varchar(50)
);
```

3. 开发：
    1. 环境搭建
        1. 创建数据库环境
        2. 创建项目，导入需要的 jar 包
    2. 编码
4. 测试
5. 部署运维

实验 9 FILTER 过滤器应用

实验目的

1. 熟悉 Filter 过滤器的原理
2. 熟悉 Filter 过滤器的使用

实验内容

Filter：过滤器

1. 概念：

* 生活中的过滤器：净水器,空气净化器，土匪、

*web 中的过滤器：当访问服务器的资源时，过滤器可以将请求拦截下来，完成一些特殊的功能。

* 过滤器的作用：

* 一般用于完成通用的操作。如：登录验证、统一编码处理、敏感字符过滤...

2. 快速入门：

1. 步骤：

1. 定义一个类，实现接口 Filter
2. 复写方法
3. 配置拦截路径

1. web.xml
2. 注解

2. 代码：

```
@WebFilter("/*")//访问所有资源之前，都会执行该过滤器
public class FilterDemo1 implements Filter {
    @Override
     public void init(FilterConfig filterConfig) throws ServletException {

    }
```

```
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("filterDemo1 被执行了....");

        //放行
filterChain.doFilter(servletRequest,servletResponse);

    }

    @Override
    public void destroy() {

    }
}
```

3. 过滤器细节：
    1. web.xml 配置

```
<filter>
    <filter-name>demo1</filter-name>
<filter-class>cn.itcast.web.filter.FilterDemo1</filter-class>
</filter>
<filter-mapping>
```

```
    <filter-name>demo1</filter-name>
    <!-- 拦截路径 -->
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. 过滤器执行流程

    1. 执行过滤器
    2. 执行放行后的资源
    3. 回来执行过滤器放行代码下边的代码

3. 过滤器生命周期方法

    1. init:在服务器启动后，会创建Filter对象，然后调用init方法。只执行一次。用于加载资源
    2. doFilter:每一次请求被拦截资源时，会执行。执行多次
    3. destroy:在服务器关闭后，Filter对象被销毁。如果服务器是正常关闭，则会执行destroy方法。只执行一次。用于释放资源

4. 过滤器配置详解

    * 拦截路径配置：

        1. 具体资源路径： /index.jsp 只有访问index.jsp资源时，过滤器才会被执行
        2. 拦截目录： /user/* 访问/user下的所有资源时，过滤器都会被执行
        3. 后缀名拦截： *.jsp 访问所有后缀名为jsp资源时，过滤器都会被执行
        4. 拦截所有资源：/* 访问所有资源时,过滤器都会被执行

    * 拦截方式配置：资源被访问的方式

        * 注解配置：

            * 设置dispatcherTypes属性

                1. REQUEST：默认值。浏览器直接请求资源
                2. FORWARD：转发访问资源

3. INCLUDE：包含访问资源

4. ERROR：错误跳转资源

5. ASYNC：异步访问资源

* web.xml 配置

* 设置<dispatcher></dispatcher>标签即可

5. 过滤器链(配置多个过滤器)

* 执行顺序：如果有两个过滤器：过滤器 1 和过滤器 2

1. 过滤器 1

2. 过滤器 2

3. 资源执行

4. 过滤器 2

5. 过滤器 1

* 过滤器先后顺序问题：

1. 注解配置：按照类名的字符串比较规则比较，值小的先执行

* 如： AFilter 和 BFilter，AFilter 就先执行了。

2. web.xml 配置： <filter-mapping>谁定义在上边，谁先执行

4. 案例：

1. 案例 1_登录验证

* 需求：

1. 访问 day17_case 案例的资源。验证其是否登录

2. 如果登录了，则直接放行。

3. 如果没有登录，则跳转到登录页面，提示"您尚未登录，请先登录"。

2. 案例2_敏感词汇过滤

* 需求：

1. 对day17_case案例录入的数据进行敏感词汇过滤
2. 敏感词汇参考《敏感词汇.txt》
3. 如果是敏感词汇，替换为 ***

* 分析：

1. 对request对象进行增强。增强获取参数相关方法
2. 放行。传递代理对象

* 增强对象的功能：

* 设计模式：一些通用的解决固定问题的方式

1. 装饰模式
2. 代理模式

* 概念：

1. 真实对象：被代理的对象
2. 代理对象：
3. 代理模式：代理对象代理真实对象，达到增强真实对象功能的目的

* 实现方式：

1. 静态代理：有一个类文件描述代理模式
2. 动态代理：在内存中形成代理类

* 实现步骤：

1. 代理对象和真实对象实现相同的接口
2. 代理对象 = Proxy.newProxyInstance();
3. 使用代理对象调用方法。

4. 增强方法

* 增强方式：

1. 增强参数列表
2. 增强返回值类型
3. 增强方法体执行逻辑

实验 10 LISTENER 监听器的使用

实验目的

1. 熟悉 Listener 监听器的原理
2. 熟悉 Listener 监听器的使用

实验内容

Listener：监听器

* 概念：web 的三大组件之一。
    * 事件监听机制
        * 事件 ： 一件事情
        * 事件源 ： 事件发生的地方
        * 监听器 ： 一个对象
        * 注册监听：将事件、事件源、监听器绑定在一起。 当事件源上发生某个事件后，执行监听器代码

* ServletContextListener:监听 ServletContext 对象的创建和销毁
    * 方法：
        * void contextDestroyed(ServletContextEvent sce) ： ServletContext 对象被销毁之前会调用该方法
        * void contextInitialized(ServletContextEvent sce) ： ServletContext 对象创建后会调用该方法
    * 步骤：
        1. 定义一个类，实现 ServletContextListener 接口
        2. 复写方法
        3. 配置
            1. web.xml

<listener>

<listener-class>cn.itcast.web.listener.ContextLoaderListener</listene

r-class>

</listener>

* 指定初始化参数<context-param>

2. 注解：

* @WebListener

实验 11 数据库连接&JDBC

实验目的

1. 熟悉数据库连接池的使用
2. 熟悉 JDBC 的使用

实验内容

数据库连接池

1. 概念：其实就是一个容器(集合)，存放数据库连接的容器。

当系统初始化好后，容器被创建，容器中会申请一些连接对象，当用户来访问数据库时，从容器中获取连接对象，用户访问完之后，会将连接对象归还给容器。

2. 好处：
    1. 节约资源
    2. 用户访问高效

3. 实现：
    1. 标准接口：DataSource javax.sql 包下的
        1. 方法：
            * 获取连接：getConnection()
            * 归还连接：Connection.close()。如果连接对象 Connection 是从连接池中获取的，那么调用 Connection.close()方法，则不会再关闭连接了。而是归还连接

2. 一般我们不去实现它，有数据库厂商来实现
    1. C3P0：数据库连接池技术
    2. Druid：数据库连接池实现技术，由阿里巴巴提供的

4. C3P0：数据库连接池技术
    * 步骤：
        1. 导入 jar 包 (两个) c3p0-0.9.5.2.jar mchange-commons-java-0.2.12.jar ，
            * 不要忘记导入数据库驱动 jar 包
        2. 定义配置文件：
            * 名称： c3p0.properties 或者 c3p0-config.xml
            * 路径：直接将文件放在 src 目录下即可。
        3. 创建核心对象 数据库连接池对象 ComboPooledDataSource
        4. 获取连接： getConnection
    * 代码：

```
//1.创建数据库连接池对象
DataSource ds  = new ComboPooledDataSource();
//2. 获取连接对象
Connection conn = ds.getConnection();
```

5. Druid：数据库连接池实现技术，由阿里巴巴提供的
    1. 步骤：
        1. 导入 jar 包 druid-1.0.9.jar
        2. 定义配置文件：
            * 是 properties 形式的
            * 可以叫任意名称，可以放在任意目录下
        3. 加载配置文件。Properties
        4. 获取数据库连接池对象：通过工厂来来获取

DruidDataSourceFactory

5. 获取连接：getConnection

* 代码：

```
//3.加载配置文件
Properties pro = new Properties();
InputStream is = DruidDemo.class.getClassLoader().getResourceAsStream("druid.properties");
pro.load(is);
//4.获取连接池对象
DataSource ds = DruidDataSourceFactory.createDataSource(pro);
//5.获取连接
Connection conn = ds.getConnection();
```

2. 定义工具类

1. 定义一个类 JDBCUtils
2. 提供静态代码块加载配置文件，初始化连接池对象
3. 提供方法
    1. 获取连接方法：通过数据库连接池获取连接
    2. 释放资源
    3. 获取连接池的方法

* 代码：

```
public class JDBCUtils {

    //1.定义成员变量 DataSource
    private static DataSource ds ;
```

```
    static{
        try {
            //1.加载配置文件
            Properties pro = new Properties();

pro.load(JDBCUtils.class.getClassLoader().getResourceAsStream("druid.
properties"));
            //2.获取 DataSource
            ds =
DruidDataSourceFactory.createDataSource(pro);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取连接
     */
    public static Connection getConnection() throws
SQLException {
        return ds.getConnection();
    }

    /**
     * 释放资源
     */
    public static void close(Statement stmt,Connection
```

```
conn){
                    /* if(stmt != null){
                         try {
                             stmt.close();
                         } catch (SQLException e) {
                             e.printStackTrace();
                         }
                     }

                     if(conn != null){
                         try {
                             conn.close();//归还连接
                         } catch (SQLException e) {
                             e.printStackTrace();
                         }
                     }*/

                    close(null,stmt,conn);
                }


                public static void close(ResultSet rs , Statement stmt,
Connection conn){


                     if(rs != null){
                         try {
                             rs.close();
                         } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
        }


        if(stmt != null){
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if(conn != null){
            try {
                conn.close();//归还连接
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * 获取连接池方法
     */

    public static DataSource getDataSource(){
        return  ds;
    }
```

}

Spring JDBC

* Spring 框架对 JDBC 的简单封装。提供了一个 JDBCTemplate 对象简化 JDBC 的开发

* 步骤：

1. 导入 jar 包
2. 创建 JdbcTemplate 对象。依赖于数据源 DataSource
    * JdbcTemplate template = new JdbcTemplate(ds);

3. 调用 JdbcTemplate 的方法来完成 CRUD 的操作
    * update():执行 DML 语句。增、删、改语句
    * queryForMap():查询结果将结果集封装为 map 集合，将列名作为 key，将值作为 value 将这条记录封装为一个 map 集合
        * 注意：这个方法查询的结果集长度只能是 1
    * queryForList():查询结果将结果集封装为 list 集合
        * 注意：将每一条记录封装为一个 Map 集合，再将 Map 集合装载到 List 集合中
    * query():查询结果，将结果封装为 JavaBean 对象
        * query 的参数：RowMapper
            * 一般我们使用 BeanPropertyRowMapper 实现类。可以完成数据到 JavaBean 的自动封装
            * new BeanPropertyRowMapper<类型>(类型.class)
    * queryForObject：查询结果，将结果封装为对象
        * 一般用于聚合函数的查询

4. 练习：
    * 需求：

1. 修改1号数据的 salary 为 10000
2. 添加一条记录
3. 删除刚才添加的记录
4. 查询id为1的记录，将其封装为Map集合
5. 查询所有记录，将其封装为List
6. 查询所有记录，将其封装为Emp对象的List集合
7. 查询总记录数

* 代码：

```
import cn.itcast.domain.Emp;
import cn.itcast.utils.JDBCUtils;
import org.junit.Test;
import
org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;

import java.sql.Date;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;

public class JdbcTemplateDemo2 {

    //Junit单元测试，可以让方法独立执行
```

```
//1. 获取 JDBCTemplate 对象
private JdbcTemplate template = new
JdbcTemplate(JDBCUtils.getDataSource());
/**
 * 1. 修改 1 号数据的 salary 为 10000
 */
@Test
public void test1(){

    //2. 定义 sql
    String sql = "update emp set salary = 10000
where id = 1001";
    //3. 执行 sql
    int count = template.update(sql);
    System.out.println(count);
}

/**
 * 2. 添加一条记录
 */
@Test
public void test2(){
    String sql = "insert into emp(id,ename,dept_id)
values(?,?,?)";
    int count = template.update(sql, 1015, "郭靖
", 10);
    System.out.println(count);

}
```

```
    /**
     * 3.删除刚才添加的记录
     */
    @Test
    public void test3(){
        String sql = "delete from emp where id = ?";
        int count = template.update(sql, 1015);
        System.out.println(count);
    }

    /**
     * 4.查询 id 为 1001 的记录，将其封装为 Map 集合
     * 注意：这个方法查询的结果集长度只能是 1
     */
    @Test
    public void test4(){
        String sql = "select * from emp where id = ? or id = ?";
        Map<String, Object> map = template.queryForMap(sql, 1001,1002);
        System.out.println(map);
        //{id=1001, ename=孙悟空, job_id=4, mgr=1004, joindate=2000-12-17, salary=10000.00, bonus=null, dept_id=20}

    }

    /**
     * 5. 查询所有记录，将其封装为 List
```

```
     */
    @Test
    public void test5(){
        String sql = "select * from emp";
        List<Map<String, Object>> list =
template.queryForList(sql);

        for (Map<String, Object> stringObjectMap :
list) {
            System.out.println(stringObjectMap);
        }
    }

    /**
     * 6. 查询所有记录，将其封装为 Emp 对象的 List 集
合
     */

    @Test
    public void test6(){
        String sql = "select * from emp";
        List<Emp> list = template.query(sql, new
RowMapper<Emp>() {

            @Override
            public Emp mapRow(ResultSet rs, int i)
throws SQLException {
                Emp emp = new Emp();
                int id = rs.getInt("id");
```

```
                String ename = rs.getString("ename");
                int job_id = rs.getInt("job_id");
                int mgr = rs.getInt("mgr");
                Date joindate =
rs.getDate("joindate");
                double salary =
rs.getDouble("salary");
                double bonus = rs.getDouble("bonus");
                int dept_id = rs.getInt("dept_id");

                emp.setId(id);
                emp.setEname(ename);
                emp.setJob_id(job_id);
                emp.setMgr(mgr);
                emp.setJoindate(joindate);
                emp.setSalary(salary);
                emp.setBonus(bonus);
                emp.setDept_id(dept_id);

                return emp;
            }
        });


        for (Emp emp : list) {
            System.out.println(emp);
        }
    }
```

```
    /**
     * 6. 查询所有记录，将其封装为 Emp 对象的 List 集
合
     */

    @Test
    public void test6_2(){
        String sql = "select * from emp";
        List<Emp> list = template.query(sql, new
BeanPropertyRowMapper<Emp>(Emp.class));
        for (Emp emp : list) {
            System.out.println(emp);
        }
    }

    /**
     * 7. 查询总记录数
     */

    @Test
    public void test7(){
        String sql = "select count(id) from emp";
        Long total = template.queryForObject(sql,
Long.class);
        System.out.println(total);
    }

}
```

实验 12 商城项目开发

实验目的

1. 熟悉商城项目的业务架构

2. 熟悉商城项目的技术架构

3. 开发出商城项目的效果

实验内容

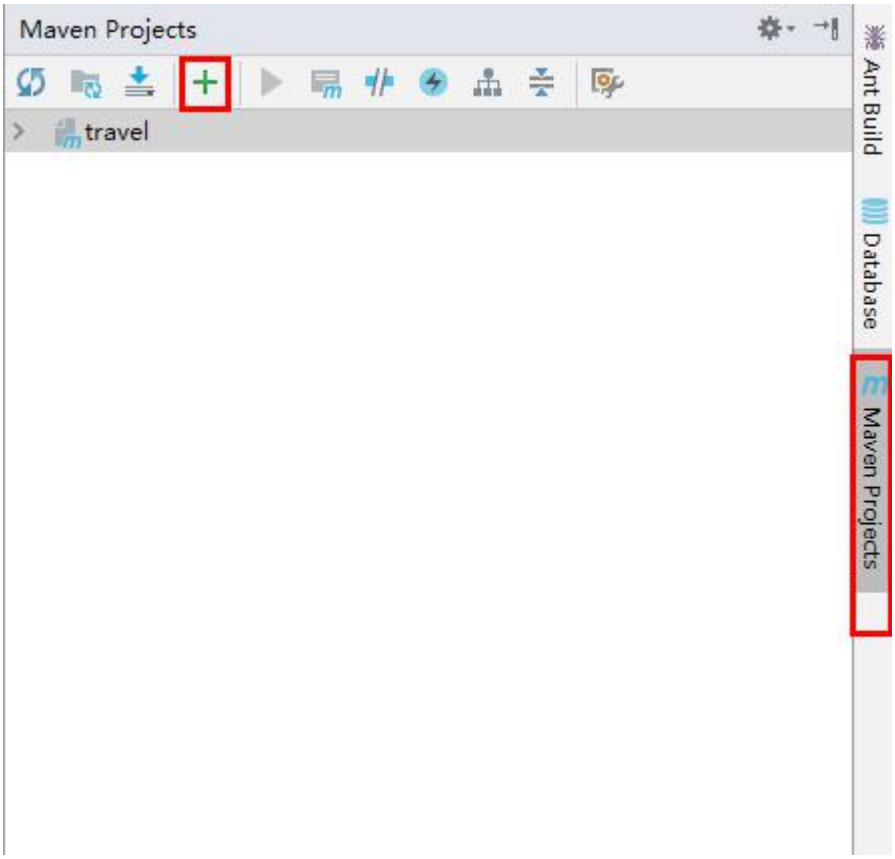
综合案例

前言

为了巩固 web 基础知识，提升综合运用能力，故而讲解此案例。要求，每位同学能够独立完成此案例。

项目导入

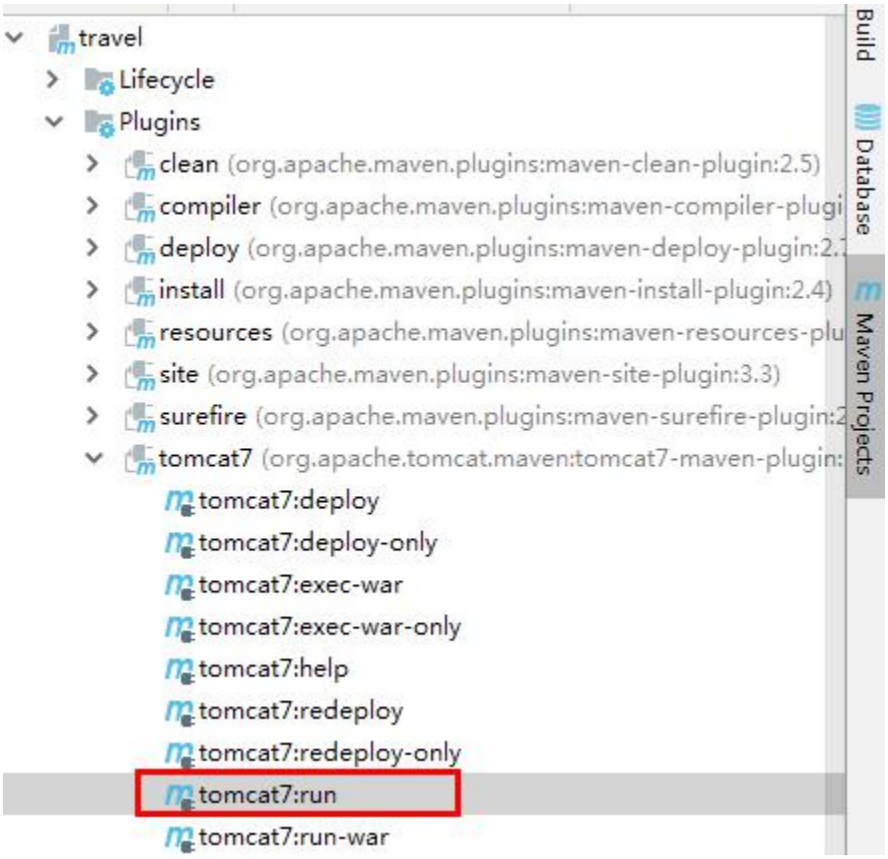
点击绿色＋按钮



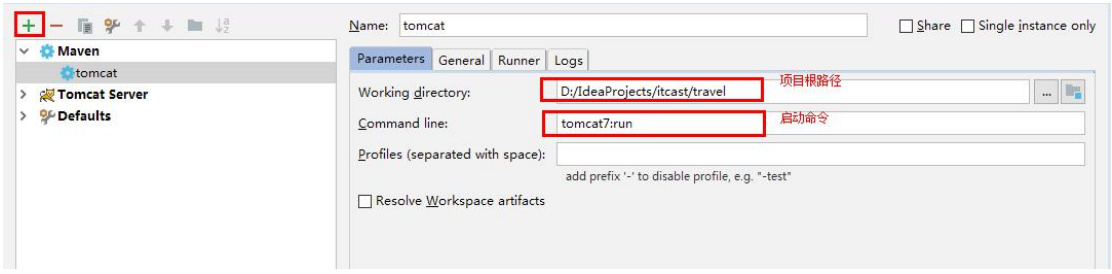
选择 travel 项目的 pom.xml 文件，点击 ok，完成项目导入。需要等待一小会，项目初始化完成。

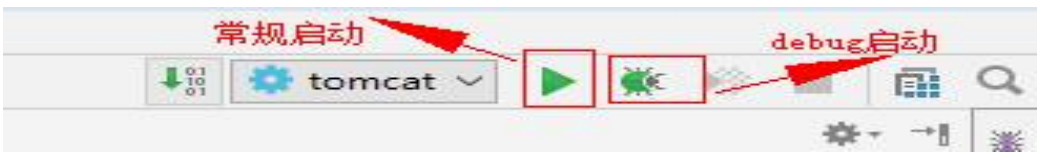
启动项目

方式一：



方式二：配置 maven 快捷启动





技术选型

Web 层

Servlet：前端控制器

html：视图

Filter：过滤器

BeanUtils：数据封装

Jackson：json 序列化工具

Service 层

Javamail：java 发送邮件工具

Redis：nosql 内存数据库

Jedis：java 的 redis 客户端

Dao 层

Mysql：数据库

Druid：数据库连接池

JdbcTemplate：jdbc 的工具

创建数据库

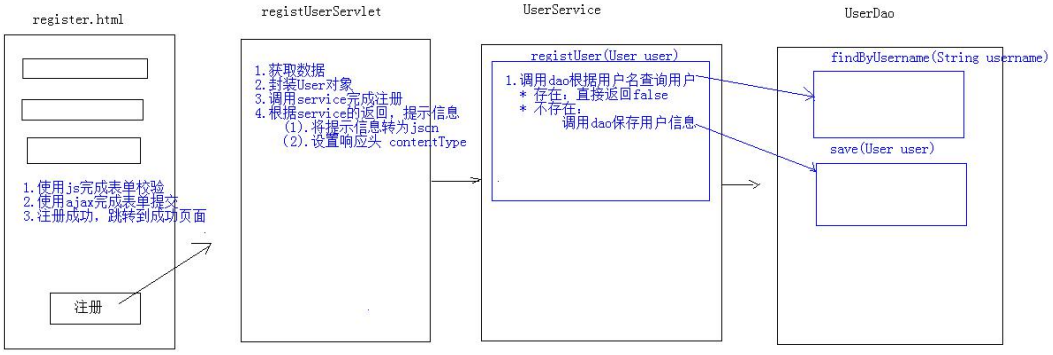
```
-- 创建数据库
CREATE DATABASE travel;
-- 使用数据库
USE travel;
--创建表
```

复制提供好的 sql

注册功能

页面效果

功能分析



代码实现

前台代码实现

表单校验

提升用户体验，并减轻服务器压力。

//校验用户名

```
//单词字符，长度8到20位
function checkUsername() {
            //1.获取用户名值
   var username = $("#username").val();
   //2.定义正则
   var reg_username = /^\w{8,20}$/;

   //3.判断，给出提示信息
    var flag = reg_username.test(username);
    if(flag){
        //用户名合法
                $("#username").css("border","");
   }else{
        //用户名非法,加一个红色边框
      $("#username").css("border","1px solid red");
   }

            return flag;
        }

        //校验密码
        function checkPassword() {
            //1.获取密码值
            var password = $("#password").val();
            //2.定义正则
            var reg_password = /^\w{8,20}$/;

            //3.判断，给出提示信息
            var flag = reg_password.test(password);
```

```
            if(flag){
                //密码合法
                $("#password").css("border","");
            }else{
                //密码非法,加一个红色边框
                $("#password").css("border","1px solid red");
            }

            return flag;
        }

        //校验邮箱
function checkEmail(){
    //1.获取邮箱
   var email = $("#email").val();
   //2.定义正则      itcast@163.com
   var reg_email = /^\w+@\w+\.\w+$/;

   //3.判断
   var flag = reg_email.test(email);
   if(flag){
                $("#email").css("border","");
   }else{
                $("#email").css("border","1px solid red");
   }

   return flag;
}
```

```
$(function () {
            //当表单提交时，调用所有的校验方法
    $("#registerForm").submit(function(){

                return checkUsername() && checkPassword() && checkEmail();
                //如果这个方法没有返回值，或者返回为 true，则表单提交，如果返回为 false，则表单不提交
    });

            //当某一个组件失去焦点是，调用对应的校验方法
        $("#username").blur(checkUsername);
            $("#password").blur(checkPassword);
            $("#email").blur(checkEmail);


        });
```

异步(ajax)提交表单

在此使用异步提交表单是为了获取服务器响应的数据。因为我们前台使用的是 html 作为视图层，不能够直接从 servlet 相关的域对象获取值，只能通过 ajax 获取响应数据

```
//当表单提交时，调用所有的校验方法
$("#registerForm").submit(function(){
    //1.发送数据到服务器
    if(checkUsername() && checkPassword() && checkEmail()){
        //校验通过,发送ajax请求，提交表单的数据   username=zhangsan&password=123
        $.post("registUserServlet",$(this).serialize(),function(data){
            //处理服务器响应的数据   data
        });

    }
    //2.不让页面跳转
    return false;
    //如果这个方法没有返回值，或者返回为true，则表单提交，如果返回为false，则表单不提交
});
```

后台代码实现

编写 RegistUserServlet

```
@WebServlet("/registUserServlet")
public class RegistUserServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {


        //验证校验
        String check = request.getParameter("check");
        //从 sesion 中获取验证码
        HttpSession session = request.getSession();
        String checkcode_server = (String) session.getAttribute("CHECKCODE_SERVER");
        session.removeAttribute("CHECKCODE_SERVER");//为了保证验证码只能使用一次
        //比较
        if(checkcode_server == null || !checkcode_server.equalsIgnoreCase(check)){
            //验证码错误
            ResultInfo info = new ResultInfo();
            //注册失败
            info.setFlag(false);
            info.setErrorMsg("验证码错误");
            //将 info 对象序列化为 json
            ObjectMapper mapper = new ObjectMapper();
            String json = mapper.writeValueAsString(info);
            response.setContentType("application/json;charset=utf-8");
            response.getWriter().write(json);
```

```
        return;
    }

    //1.获取数据
    Map<String, String[]> map = request.getParameterMap();

    //2.封装对象
    User user = new User();
    try {
        BeanUtils.populate(user,map);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    }

    //3.调用 service 完成注册
    UserService service = new UserServiceImpl();
    boolean flag = service.regist(user);
    ResultInfo info = new ResultInfo();
    //4.响应结果
    if(flag){
        //注册成功
        info.setFlag(true);
    }else{
        //注册失败
        info.setFlag(false);
        info.setErrorMsg("注册失败!");
    }
```

```
        //将 info 对象序列化为 json
        ObjectMapper mapper = new ObjectMapper();
        String json = mapper.writeValueAsString(info);

        //将 json 数据写回客户端
        //设置 content-type
        response.setContentType("application/json;charset=utf-8");
        response.getWriter().write(json);


    }

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

编写 UserService 以及 UserServiceImpl

```
public class UserServiceImpl implements UserService {

private UserDao userDao = new UserDaoImpl();
/**
 * 注册用户
 * @param user
 * @return
 */
@Override
```

```
    public boolean regist(User user) {
        //1.根据用户名查询用户对象
        User u = userDao.findByUsername(user.getUsername());
        //判断 u 是否为 null
        if(u != null){
            //用户名存在，注册失败
            return false;
        }
        //2.保存用户信息
        userDao.save(user);
        return true;
    }
}
```

编写 UserDao 以及 UserDaoImpl

```
    public class UserDaoImpl implements UserDao {

    private JdbcTemplate template = new
JdbcTemplate(JDBCUtils.getDataSource());

    @Override
    public User findByUsername(String username) {
        User user = null;
        try {
            //1.定义 sql
            String sql = "select * from tab_user where username = ?";
            //2.执行 sql
            user = template.queryForObject(sql, new
```

```
BeanPropertyRowMapper<User>(User.class), username);
        } catch (Exception e) {

        }

        return user;
    }

    @Override
    public void save(User user) {
        //1.定义 sql
        String sql = "insert into tab_user(username,password,name,birthday,sex,telephone,email) values(?,?,?,?,?,?,?)";
        //2.执行 sql

        template.update(sql,user.getUsername(),
                    user.getPassword(),
                user.getName(),
                user.getBirthday(),
                user.getSex(),
                user.getTelephone(),
                user.getEmail());
    }
}
```

邮件激活

为什么要进行邮件激活？为了保证用户填写的邮箱是正确的。将来可以推广一些宣传信息，到用户邮箱中。

发送邮件

申请邮箱

开启授权码

在 MailUtils 中设置自己的邮箱账号和密码(授权码)

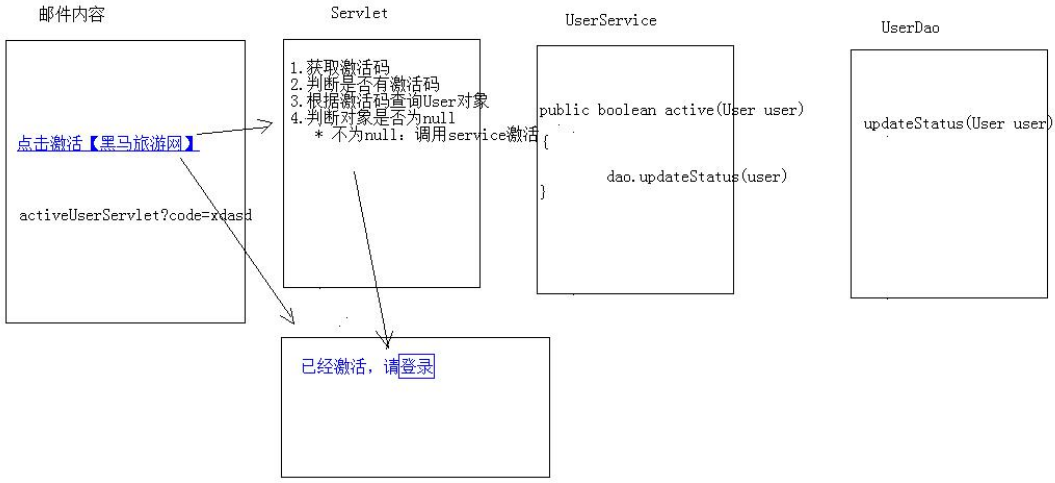
邮件工具类：MailUtils，调用其中 sendMail 方法可以完成邮件发送

用户点击邮件激活

经过分析，发现，用户激活其实就是修改用户表中的 status 为‘Y’

```
public class User implements Serializable {
    private int uid;//用户id
    private String username;//用户名，账号
    private String password;//密码
    private String name;//真实姓名
    private String birthday;//出生日期
    private String sex;//男或女
    private String telephone;//手机号
    private String email;//邮箱
    private String status;//激活状态，Y代表激活，N代表未激活
    private String code;//激活码（要求唯一）
```

分析：



发送邮件代码：

```
@Override
public boolean regist(User user) {
    //1.根据用户名查询用户对象
    User u = userDao.findByUsername(user.getUsername());
    //判断u是否为null
    if(u != null){
        //用户名存在，注册失败
        return false;
    }
    //2.保存用户信息
    //2.1设置激活码，唯一字符串
    user.setCode(UuidUtil.getUuid());
    //2.2设置激活状态
    user.setStatus("N");
    userDao.save(user);

    //3.激活邮件发送，邮件正文?

    String content="<a href='http://localhost/travel/activeUserServlet?code="+user.getCode(

    MailUtils.sendMail(user.getEmail(),content, title: "激活邮件");

    return true;
}
```

修改保存 Dao 代码，加上存储 status 和 code 的代码逻辑

```
@Override
public void save(User user) {
    //1.定义sql
    String sql = "insert into tab_user(username,password,name,birthday,sex,telephone,email,status,code) values(?,?,?,?,?,?,?,?,?)";
    //2.执行sql

    template.update(sql,user.getUsername(),
                user.getPassword(),
            user.getName(),
            user.getBirthday(),
            user.getSex(),
            user.getTelephone(),
            user.getEmail(),
            user.getStatus(),
            user.getCode()
            );
}
```

激活代码实现：

ActiveUserServlet

```
//1.获取激活码
String code = request.getParameter("code");
if(code != null){
    //2.调用 service 完成激活
    UserService service = new UserServiceImpl();
    boolean flag = service.active(code);

    //3.判断标记
    String msg = null;
    if(flag){
        //激活成功
        msg = "激活成功，请<a href='login.html'>登录</a>";
    }else{
        //激活失败
        msg = "激活失败，请联系管理员!";
    }
    response.setContentType("text/html;charset=utf-8");
    response.getWriter().write(msg);
```

UserService：active

```
    @Override
public boolean active(String code) {
    //1.根据激活码查询用户对象
    User user = userDao.findByCode(code);
    if(user != null){
        //2.调用 dao 的修改激活状态的方法
        userDao.updateStatus(user);
```

```
        return true;
    }else{
        return false;
    }

}
```

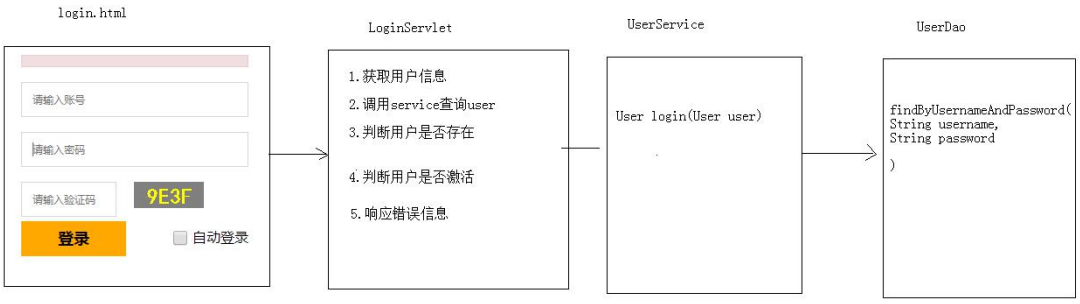
UserDao: findByCode updateStatus

```
    /**
 * 根据激活码查询用户对象
 * @param code
 * @return
 */
@Override
public User findByCode(String code) {
    User user = null;
    try {
        String sql = "select * from tab_user where code = ?";

        user            =            template.queryForObject(sql,new
BeanPropertyRowMapper<User>(User.class),code);
    } catch (DataAccessException e) {
        e.printStackTrace();
    }

    return user;
}
```

```
/**
 * 修改指定用户激活状态
 * @param user
 */
@Override
public void updateStatus(User user) {
    String sql = " update tab_user set status = 'Y' where uid=?";
    template.update(sql,user.getUid());
}
```

登录

分析



代码实现

前台代码

```
$(function () {
    //1.给登录按钮绑定单击事件
    $("#btn_sub").click(function () {
        //2.发送ajax请求，提交表单数据
        $.post("loginServlet",$("#loginForm").serialize(),function (data) {
            //data : {flag:false,errorMsg:''}
            if(data.flag){
                //登录成功
                location.href="index.html";
            }else{
                //登录失败
                $("#errorMsg").html(data.errorMsg);
            }
        });
    });
});
```

后台代码

LoginServlet

```
//1.获取用户名和密码数据
Map<String, String[]> map = request.getParameterMap();
//2.封装 User 对象
User user = new User();
try {
    BeanUtils.populate(user,map);
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}

//3.调用 Service 查询
UserService service = new UserServiceImpl();
User u  = service.login(user);

ResultInfo info = new ResultInfo();

//4.判断用户对象是否为 null
if(u == null){
```

```
    //用户名密码或错误
    info.setFlag(false);
    info.setErrorMsg("用户名密码或错误");
}
//5.判断用户是否激活
if(u != null && !"Y".equals(u.getStatus())){
    //用户尚未激活
    info.setFlag(false);
    info.setErrorMsg("您尚未激活，请激活");
}
//6.判断登录成功
if(u != null && "Y".equals(u.getStatus())){
    //登录成功
    info.setFlag(true);
}

//响应数据
ObjectMapper mapper = new ObjectMapper();

response.setContentType("application/json;charset=utf-8");
mapper.writeValue(response.getOutputStream(),info);

    UserService
    public User login(User user) {
    return
userDao.findByUsernameAndPassword(user.getUsername(),user.getPassword
());
}
```

UserDao

```
public User findByUsernameAndPassword(String username, String password) {
    User user = null;
    try {
        //1.定义 sql
        String sql = "select * from tab_user where username = ? and password = ?";
        //2.执行 sql
        user = template.queryForObject(sql, new BeanPropertyRowMapper<User>(User.class), username,password);
    } catch (Exception e) {

    }

    return user;
}
```

index 页面中用户姓名的提示信息功能

效果:

header.html 代码

```
$(function () {
    $.get("findUserServlet",{},function (data) {
        //{uid:1,name:'李四'}
        var msg = "欢迎回来，"+data.name;
        $("#span_username").html(msg);

    });
})
```

Servlet 代码

```
//从 session 中获取登录用户
Object user = request.getSession().getAttribute("user");
```

```
//将 user 写回客户端

ObjectMapper mapper = new ObjectMapper();
response.setContentType("application/json;charset=utf-8");
mapper.writeValue(response.getOutputStream(),user);
```

退出

什么叫做登录了？session 中有 user 对象。

实现步骤：

访问 servlet，将 session 销毁

跳转到登录页面

代码实现：

Header.html

```
<div class="login">

    <span id="span_username"></span>
    <a href="myfavorite.html" class="collection">我的收藏</a>
    <a href="javascript:location.href='exitServlet';">退出</a>
</div>
```

Servlet:

```
//1.销毁 session
request.getSession().invalidate();

//2.跳转登录页面
response.sendRedirect(request.getContextPath()+"/login.html");
```