

web核心--综合案例

学习目标

1. 能够使用mvc及三层实现用户的列表查询操作
2. 能够使用mvc及三层实现用户的新增操作
3. 能够使用mvc及三层实现用户的修改操作
4. 能够使用mvc及三层实现用户的删除操作
5. 能够使用ajax实现验证用户名是否存在
6. 能够使用fileupload实现文件上传

第一章 用户管理CRUD操作

1. 项目概述

黑马旅游本着以“让旅游更简单”为使命，为消费者提供由各大城市出发的旅游产品预订服务，产品全面，价格透明，全年365天24小时400电话预订，并提供丰富的后续服务和保障。

其中用户信息管理针对用户的大量业务处理工作而开发的管理软件，主要用于本网站注册用户的信息管理，总体任务是实现用户信息的系统化、科学化、规范化和自动化，其主要任务是用计算机对用户各种信息进行日常管理，如查询、修改、增加、删除等功能。


2. 项目案例需求

我们本次课程的目标就是完成对用户信息的管理，实现用户信息的增、删、改、查操作。





3. 项目环境搭建

资料：

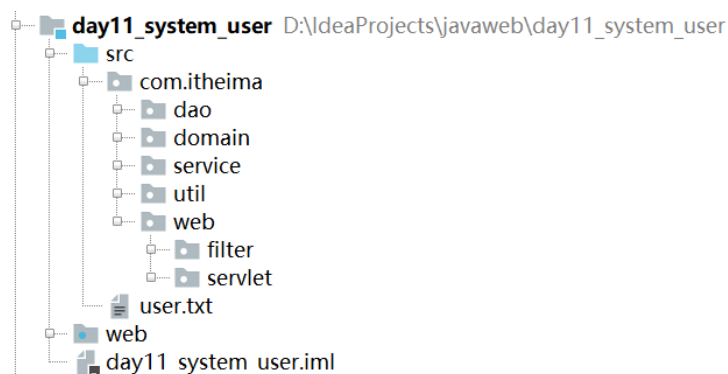
拷贝 day07_综合 案例\资料 下的文件

- | | |
|--|-------------|
|  jar包 | 项目中所需的jar文件 |
|  DataUtils.java | 读写文件的工具类 |
|  user.txt | 保存用户信息的数据文件 |

导入的jar包 说明：

- | | |
|---|------------------|
|  commons-beanutils-1.8.3.jar | 封装请求参数使用的工具包及日志包 |
|  commons-logging-1.1.1.jar | |
|  javax.servlet.jsp.jstl.jar | jstl标签库使用的相关jar包 |
|  jstl-impl.jar | |

项目结构：

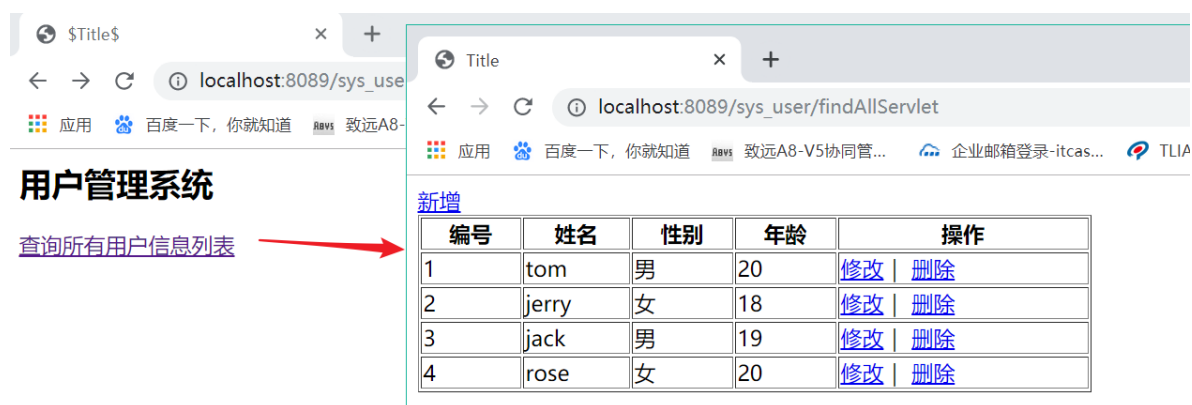


4. 查询所有用户信息

需求：

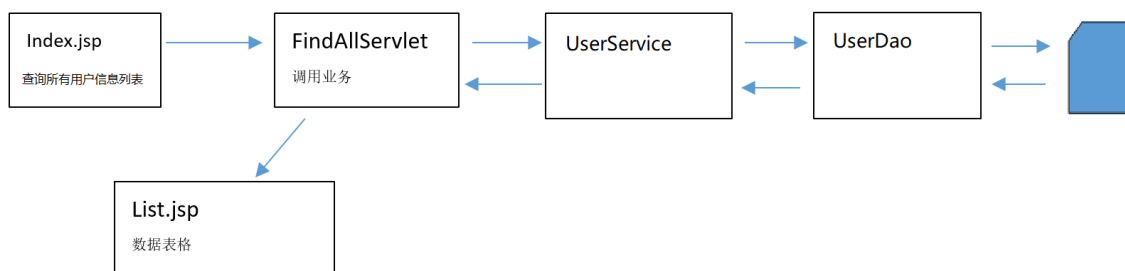
当程序运行时，访问网站首页，点击“查询所有用户信息列表”超链接，跳转至用户列表页面展示用户信息列表。

如下图所示：



思路流程分析：

首先我们访问index.jsp页面，当点击“查询”按钮时，程序应跳转到FindAllServlet，目标是通过FindAllServlet向页面返回一个用户的list集合，这样才能在list.jsp页面中有可以遍历的数据；再通过FindAllServlet调用UserService业务代码，业务代码继续调用UserDao，则UserDao实现文件中的数据查询，并返回list集合数据。执行流程如下图所示：



创建index.jsp

```
<body>
    <h2>用户管理系统</h2>
    <a href="${pageContext.request.contextPath}/findAllServlet">查询所有用户信息列表</a>
</body>
```

创建FindAllServlet

```
@WebServlet(urlPatterns = "/findAllServlet")
public class FindAllServlet extends HttpServlet{

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //调用业务逻辑
        UserService userService = new UserService();
        List<User> list = userService.findAll();

        //分发转向
        req.setAttribute("list", list); //保存集合数据到request域对象中
        //因为需要传递数据到下一个资源，所以要使用转发
        req.getRequestDispatcher("/list.jsp").forward(req, resp);
    }

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

创建UserService

```
public class UserService {

    public List<User> findAll() {
        UserDao userDao = new UserDao();
        return userDao.findAll();
    }
}
```

创建UserDao

```

public class UserDao {

    //查询所有
    public List<User> findAll(){
        DataUtil dataUtil = new DataUtil();
        return dataUtil.readAll(); //从文件中读取封装到集合的数据
    }
}

```

创建list.jsp

```

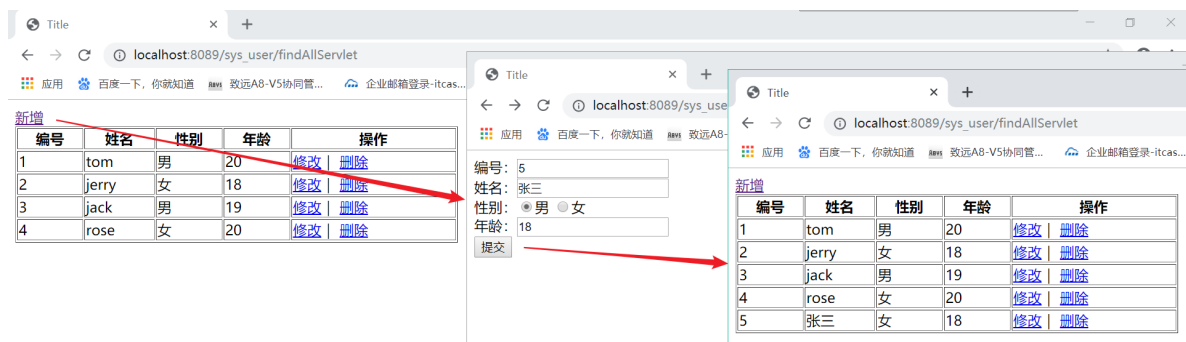
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <table border="1" width="500">
        <tr>
            <th>编号</th>
            <th>姓名</th>
            <th>性别</th>
            <th>年龄</th>
            <th>操作</th>
        </tr>
        <c:forEach var="user" items="${list}">
            <tr>
                <td>${user.id}</td>
                <td>${user.name}</td>
                <td>${user.gender=="m"? "男": "女"}</td>
                <td>${user.age}</td>
                <td>
                    <a href="">修改</a> |
                    <a href="">删除</a>
                </td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>

```

5. 新增用户

需求：

当用户点击“新增”按钮，跳转到新增页面，用户输入信息后，点击“提交”按钮，保存数据到文件中，然后再跳转到列表页面，展示用户列表信息。如下图所示：

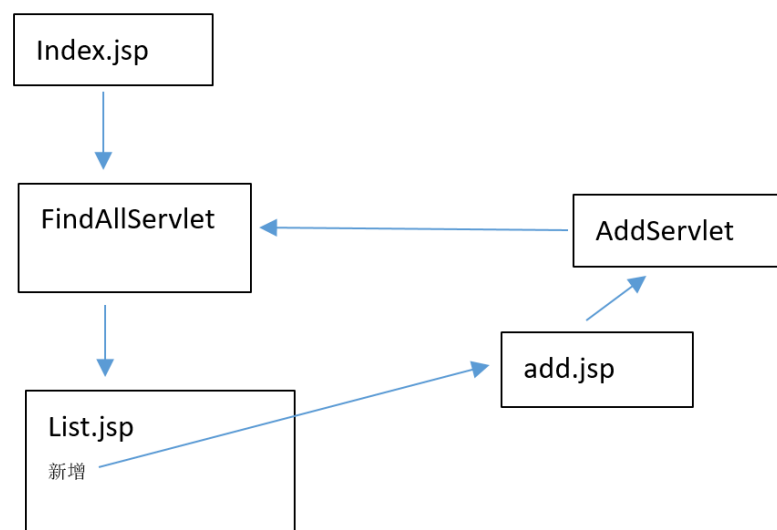


思路流程分析：

当用户点击list.jsp页面中的“新增”按钮后，要给用户手动添加数据的机会，所以要跳转到add.jsp页面；用户输入完数据后，点击“提交”按钮，要跳转至AddServlet接收页面提交的数据，然后依次调用UserService和UserDao，并保存数据到文件中；

需要注意的是，当AddServlet跳转时,如果直接跳转到list.jsp页面，则页面中不会有任何数据显示，正确流程是应该跳转到FindAllServlet，由FindAllServlet转发到list.jsp页面就会展示新的列表数据了，也就是说此时FindAllServlet重新查询了一次列表数据。

注：因为所有流程都需要调用业务层和dao层，由于篇幅有限下图中省略了Service和Dao部分图解



在list.jsp页面中添加一个“新增”按钮

```
<a href="{pageContext.request.contextPath}/add.jsp">新增</a>
```

创建add.jsp

```
<form action="${pageContext.request.contextPath}/addServlet" method="post">
    编号: <input type="text" name="id"/><br/>
    姓名: <input type="text" name="name"/><br/>
    性别: <input type="radio" name="gender" value="男" checked/>男
    <input type="radio" name="gender" value="女"/>女<br/>
    年龄: <input type="text" name="age"/><br/>
    <input type="submit" value="提交"/>
</form>
```

创建AddServlet

```
package com.itheima.web.servlet;

@WebServlet(urlPatterns = "/addServlet")
public class AddServlet extends HttpServlet{

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

        try {
            //获取请求参数
            User user = new User();
            BeanUtils.populate(user, req.getParameterMap());
            //调用业务逻辑
            UserService userService = new UserService();
            userService.add(user);
            //分发转向
            response.sendRedirect("/findAllServlet");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

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

修改UserService

```
public class UserService {

    public void add(User user) {
        UserDao userDao = new UserDao();
        userDao.add(user);
    }
}
```

修改 UserDao

```
//添加用户
public void add(User user){
    DataUtils dataUtil = new DataUtils();
    List<User> list = dataUtil.readAll();
    list.add(user);
    dataUtil.writeAll(list); //写入集合数据到文件中
}
```

6. 解决全站乱码

当添加用户功能完成后，发现列表中的用户信息乱码了，接下来我们利用学习过的过滤器来解决一下全站的中文乱码问题。

```
package com.itheima.web.filter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebFilter(urlPatterns = "/*")
public class CodingFilter implements Filter{
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    //过滤
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest request;
        HttpServletResponse response;
        try {
            request = (HttpServletRequest)servletRequest;
            response = (HttpServletResponse)servletResponse;
        } catch (ClassCastException var6) {
            throw new ServletException("non-HTTP request or response");
        }
        //处理post请求乱码
        request.setCharacterEncoding("UTF-8");
        //处理响应乱码
        response.setContentType("text/html;charset=UTF-8");

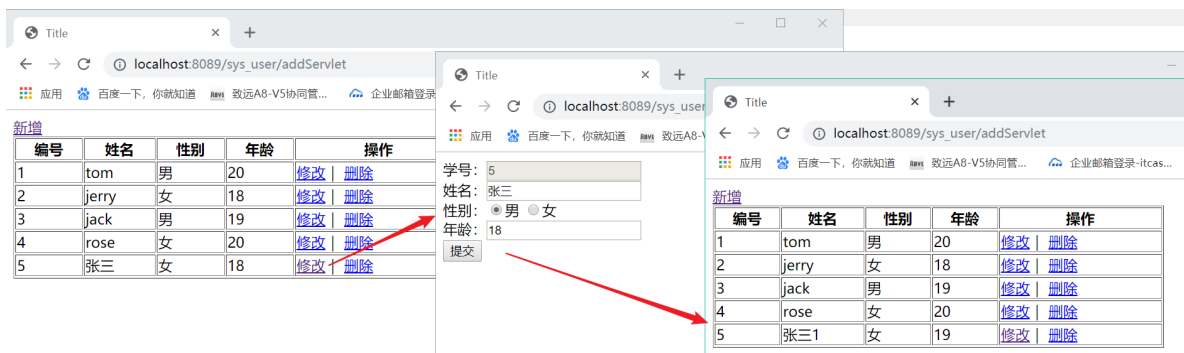
        //放行
        filterChain.doFilter(request,response);
    }
}
```

```
public void destroy() {  
  
}  
}
```

7. 更新用户信息

需求：

点击“修改”按钮，跳转至修改用户信息页面，并回显要被修改的用户信息，点击“提交”按钮后，再次跳转到列表页面，展示修改后的用户信息。效果如下图所示：



思路流程分析：

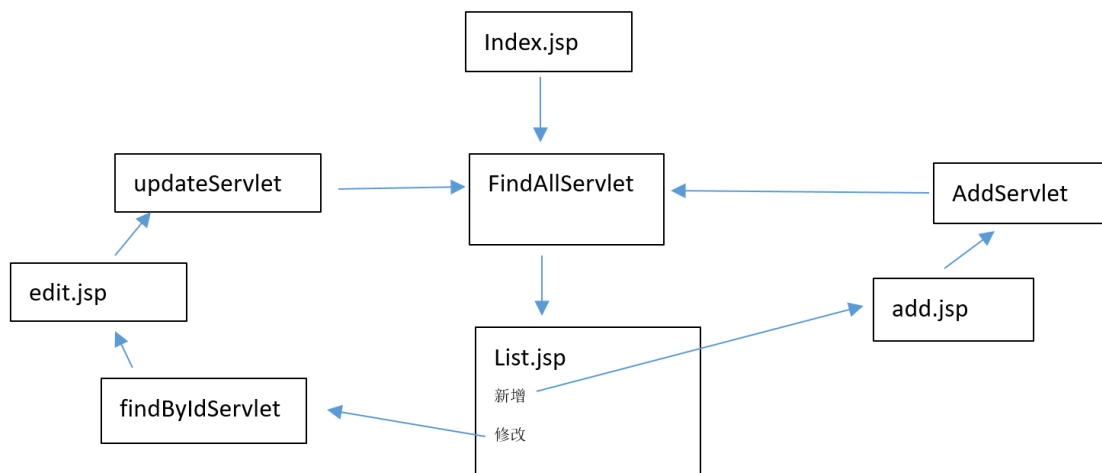
修改功能的流程相对稍麻烦一点，修改功能整个流程需要经过3个Servlet，不过虽然麻烦一点，但程序的流程也不难理解，我们只需要认真体会就比较容易做到了。

我们点击“修改”按钮要跳转到修改页面，为了用户使用体验更好些，不能让用户手动再输入一遍表单中的用户数据，所以，在点击“修改”超链接时，需要传递当前用户id到findByIdServlet，然后Servlet中根据用户id查询到用户对象，再转发到edit.jsp页面就可以回显用户数据了；

当修改用户数据页面点击“提交”表单，跳转到updateServlet，这时调用业务层和dao层代码，执行更新文件中的数据操作；

更新完成后，updateServlet要跳转到FindAllServlet完成重新查询用户集合数据，列表页面中就可以显示最新用户列表数据了。

执行流程如下图所示：



修改list.jsp的路径

```
<a href="${pageContext.request.contextPath}/findByIdServlet?id=${user.id}">修改  
</a> |
```

创建FindByIdServlet

```
package com.itheima.web.servlet;

@WebServlet(urlPatterns = "/findByIdServlet")
public class FindByIdServlet extends HttpServlet{

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //获取请求参数
        String id = req.getParameter("id");

        //调用业务逻辑
        UserService userService = new UserService();
        User user = userService.findById(id);
        //分发转向
        req.setAttribute("user",user);
        req.getRequestDispatcher("/edit.jsp").forward(req,resp);
    }

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

创建edit.jsp页面

```
<form action="${pageContext.request.contextPath}/updateServlet" method="post">
    学号: <input type="text" disabled name="id" value="${user.id}"/><br/>
    姓名: <input type="text" name="name" value="${user.name}"/><br/>
    性别: <input type="radio" name="gender" value="男"
    ${user.gender=="男"?checked:""}/>男
        <input type="radio" name="gender" value="女"
    ${user.gender=="女"?checked:""}/>女<br/>
    年龄: <input type="text" name="age" value="${user.age}"/><br/>
    <input type="submit" value="提交"/>
</form>
```

创建UpdateServlet

```
package com.itheima.web.servlet;

@WebServlet(urlPatterns = "/updateServlet")
public class UpdateServlet extends HttpServlet{

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

        try {
            //获取请求参数
            User user = new User();
            BeanUtils.populate(user, req.getParameterMap());
            //调用业务逻辑
            UserService userService = new UserService();
            userService.update(user);

            //分发转向
            req.getRequestDispatcher("/findAllServlet").forward(req, resp);

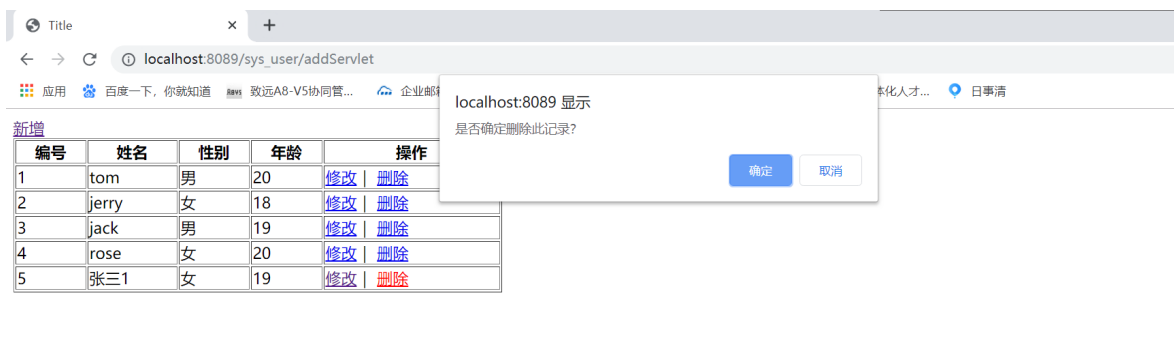
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

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

8. 删除用户

需求:

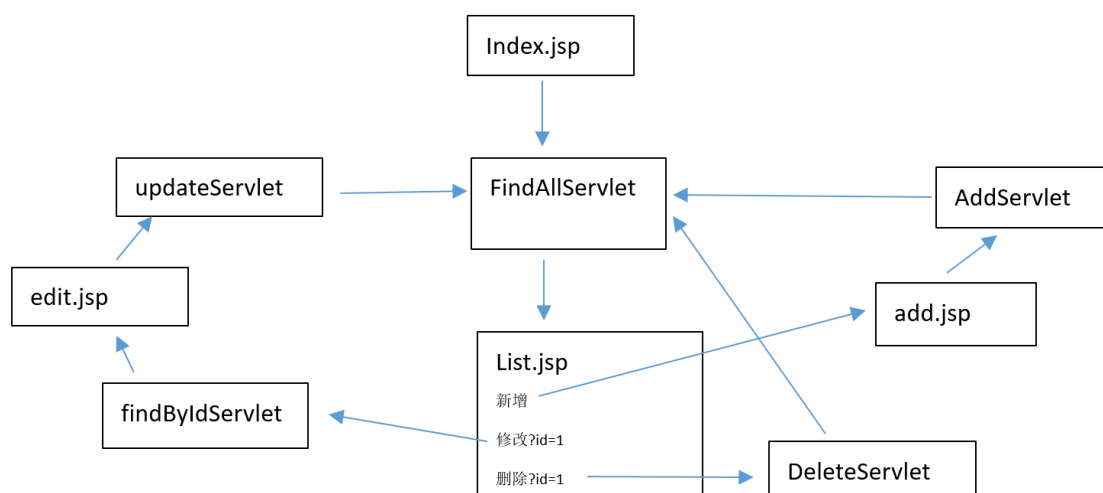
点击“删除”超链接，删除指定的用户信息，删除要给用户提示确认。



思路流程分析：

删除用户可以说是最简单的一个流程了，我们只需要向后台代码传递一个用户编号就可以删除指定的用户了。当点击“删除”超链接，需要传递用户id到DeleteServlet，再调用业务层和dao层代码，执行删除操作；然后也是跳转到FindAllServlet就完成了整个删除操作的流程。

这里需要注意的是，点击“删除”超链接，要调用javascript代码跳转DeleteServlet才可以实现删除提示功能。具体操作请参见后面的示例代码。



修改list.jsp

```
<script>
function deleteById(id) {
    //confirm是带有“确定”和“取消”按钮的提示框，返回值是boolean类型
    if(confirm("是否确定删除此记录? ")){
        location.href= "${pageContext.request.contextPath}/deleteByIdServlet?
id="+id;
    }
}
</script>

<a href="javascript:deleteById('${user.id}')">删除</a>
```

创建DeleteByIdServlet

```
package com.itheima.web.servlet;

@WebServlet(urlPatterns = "/deleteByIdServlet")
public class DeleteByIdServlet extends HttpServlet{

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //获取请求参数
        String id = req.getParameter("id");
        //调用业务逻辑
        UserService userService = new UserService();
        userService.deleteById(id);
        //分发转向
        response.sendRedirect("/findAllServlet");
    }

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

修改UserService

```
public void deleteById(String id) {
    UserDao userDao = new UserDao();
    userDao.deleteById(id);
}
```

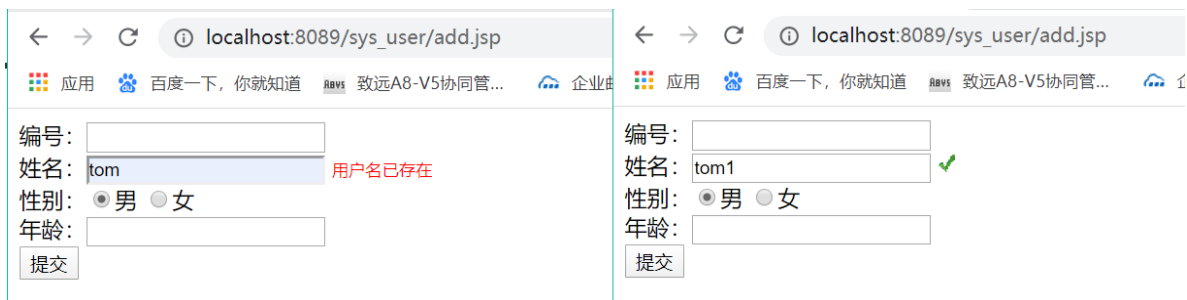
修改UserDao

```
//删除用户
public void deleteById(String id){
    DataUtils dataUtil = new DataUtils();
    List<User> list = dataUtil.readAll();
    for (int i = 0; i < list.size(); i++) {
        User user = list.get(i);
        if(user.getId().equals(id)){
            list.remove(user);
        }
    }
    dataUtil.writeAll(list); //写入集合数据到文件中
}
```

第二章 ajax验证用户名是否存在

1、需求：

在页面不刷新的情况下，实现验证用户名是否存在。效果如下图：



2、思路分析：

当用户输入完用户名，光标离开焦点后，使用ajax向后台服务器发送请求，并传递用户名。后台接收用户名，判断用户名是否已存在，并响应结果。

3、代码实现

编写add.jsp页面

```
<%@ page import="java.util.Random" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <style>
        .msg{
            font-size:12px;
            color:red;
        }
    </style>
    <script src="js/jquery-1.11.3.js"></script>
    <script>
        //页面加载事件
        $(function () {
            //获得"姓名"文本框，并绑定失去焦点事件。
            $("input[name='name']").blur(function () {
                //定义json格式的参数字
                var param = {"name": $(this).val()};
                //发送ajax，验证用户名是否存在。
                $.get("${pageContext.request.contextPath}/checkNameServlet",
                    param, function (result) {
                        if(result=="y"){
```

```

        //向span标签中添加文本，并调用类样式
        $("#msg").html("用户名已存在").addClass("msg");
    }else{
        //向span标签中添加图片
        $("#msg").html("<img src='images/y.png' height='14'
width='16' />");
    }
    });
});
});
</script>
</head>
<body>

<form action="${pageContext.request.contextPath}/addServlet" method="post">
    编号: <input type="text" name="id"/><br/>
    姓名: <input type="text" name="name"/> <span id="msg"></span><br/>
    性别: <input type="radio" name="gender" value="男" checked/>男
    <input type="radio" name="gender" value="女"/>女<br/>
    年龄: <input type="text" name="age"/><br/>
    <input type="submit" value="提交"/>
</form>
</body>
</html>

```

创建CheckNameServlet

```

@WebServlet(urlPatterns = "/checkNameServlet")
public class CheckNameServlet extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        //获取请求参数
        String name = req.getParameter("name");

        //调用业务逻辑
        UserService userService = new UserService();

        User user = userService.findByName(name);

        //响应结果
        if(user!=null){
            resp.getWriter().write("y");
        }else{
            resp.getWriter().write("n");
        }
    }

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

```

修改UserService

```
public Student findByName(String name) {  
    UserDao userDao = new UserDao();  
    return userDao.findByName(name);  
}
```

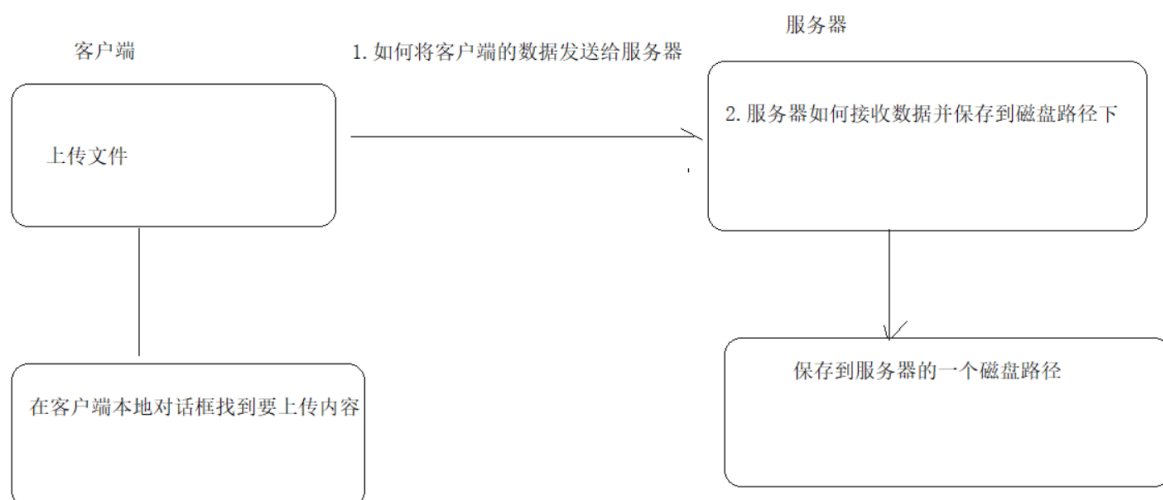
修改UserDao

```
public User findByName(String name) {  
    DataUtils dataUtil = new DataUtils();  
    List<User> list = dataUtil.readAll();  
    for (User user : list) {  
        if(user.getName().equals(name)){  
            return user;  
        }  
    }  
    return null;  
}
```

第三章 文件上传

什么是文件上传？

客户端将本地的文件传递到服务器的过程，就称之为文件上传。



1. 实现文件上传的三要素

1. 必须使用post请求

```
<form action="..." method="post"></form>
```

2. file标签必须有name属性

```
<input type = "file" name="..." />
```

3. form的enctype必须登录multipart/form-data

```
<form action="..." method="post" enctype="multipart/form-data"></form>
```

默认值 普通的表单提交

```
enctype="application/x-www-form-urlencoded"
```

完整

```
<form action="..." method="post" enctype="multipart/form-data">
  <input type="file" name=".." />
</form>
```

```
<body>
```

```
<form action="#" method="post" enctype="multipart/form-data"> 表单分成多部分提交
```

文件描述: <input type="text" name="desc">

文件: <input type="file" name="fi">


```
<input type="submit" value="Submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

上传流程

客户端在提交请求时 增加了 multipart/form-data选择(多出了一个分割线内容)

服务器在解析时, 会按照分割线来对请求体进行解析

普通项(获取字符串即可) 和 上传项(获取IO流)的解析方式不同

请求头 (662 字节)

```
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
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
Content-Type: multipart/form-data; boundary=-----191691572411478
Content-Length: 556
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/sys_student/fileupload/upload.jsp
Cookie: JSESSIONID=02727018E9B26903920225BA96C69594
Upgrade-Insecure-Requests: 1
```

Tomcat 服务器

获取提交方式: multipart/form-data
读取分割线
boundary=-----191691572411478

有效载荷 (payload)

```
-----191691572411478
Content-Disposition: form-data; name="desc"

aaa
bbb
ccc
ddd
-----191691572411478
Content-Disposition: form-data; name="fi"; filename="a.txt"
Content-Type: text/plain

aaa
bbb
ccc
ddd
-----191691572411478
Content-Disposition: form-data; name="desc"

aaa
bbb
ccc
ddd
-----191691572411478
Content-Disposition: form-data; name="fi2"; filename="b.txt"
Content-Type: text/plain

xxx
yyy
zzz
-----191691572411478
```

解析请求体: 按照分割线来解析

判断当前的请求体是普通项(字符串) 还是 上传项(IO)

普通项和上传项区别
上传项多出了 filename 字段

分割线: 设置完enctype之后产生的分割线

有效载荷 (payload)

```
-----18467633426500
Content-Disposition: form-data; name="desc"

波波
-----18467633426500
Content-Disposition: form-data; name="fi"; filename="a.txt"
Content-Type: text/plain

aaa
bbb
ccc
ddd
-----18467633426500--
```

普通项

参数名

参数值

上传项

文件名

文件

2. 文件上传的几种方式

1. Apache提供的一个 commons-fileupload 包实现[今天讲解]（最麻烦）
2. Servlet3.0提供的 注解方式上传文件，很简单
3. SpringMVC实现的文件上传，（简单的有点过分）

导入依赖包

lib

> commons-fileupload-1.3.3.jar

> commons-io-2.6.jar

> javax.servlet.jsp.jstl.jar

> jstl-impl.jar

web.xml

实现文件上传的解析功能

实现了IO操作的功能

protected void

// 实现-

// 1. 创

// 2. 创

ContentType: multipart/form-data作用

请求头 (651 字节)

Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
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
Content-Type: multipart/form-data; boundary=-----57052814523281
Content-Length: 301
Origin: http://localhost:8080
Connection: keep-alive
Referer: http://localhost:8080/sys_student/fileupload/upload.jsp
Cookie: JSESSIONID=AD1A680AB125AE1735E9781CFB98E1F4
Upgrade-Insecure-Requests: 1

multipart/form-data: 将请求的数据分成多个块
boundary: 分割线, 分割请求用

请求分段

请求有效载荷 (payload)

1 -----57052814523281
2 Content-Disposition: form-data; name="desc"
3
4 谈谈
5 -----57052814523281
6 Content-Disposition: form-data; name="fi"; filename="a.txt"
7 Content-Type: text/plain
8
9 aaa
0 bbb
1 ccc
2 ddd
3 -----57052814523281--
4 结束

使用分割线对请求提交的
数据分称多个段
普通项和上传项
普通项和上传项的区别: 是否有filename字段

实现文件上传的jsp

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        <form action="${pageContext.request.contextPath}/file/upload"
method="post" enctype="multipart/form-data">
            文件描述: <input type="text" name="desc"><br/>
            文件: <input type="file" name="fi"><br/>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>

```

实现文件上传的Servlet

```

// 实现文件上传功能
@WebServlet(urlPatterns = "/file/upload")
public class FileUploadServlet extends HttpServlet {

    // 1. 必须为post请求
    // 2. file标签必须有name
    // 3. enctype = multipart/form-data

    // 使用 commons-fileupload 实现文件上传的步骤是固定的
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 实现一个文件上传功能的步骤
        // 1. 创建磁盘操作工具类工厂 工厂：各种创建对象 创建磁盘操作的对象
        DiskFileItemFactory factory = new DiskFileItemFactory(); // commons-
fileupload提供的工具类
        // 2. 创建核心解析类
        ServletFileUpload sfu = new ServletFileUpload(factory);
        // 3. 解析Request请求 返回list结合 -> 分割线的内容
        try {
            List<FileItem> list = sfu.parseRequest(req); // List<FileItem> : 表单
提交的各种块
            for (FileItem fileItem : list) { // 迭代 上传块
                if (fileItem.isFormField()) { // 已经封装好的 函数 true: 普通项
                    // 普通项有两个主要属性 参数名, 参数值
                    String fieldName = fileItem.getFieldName(); // 参数名
                    String value = fileItem.getString(); // 参数值
                    System.out.println(fieldName + " " + value);
                } else {
                    // 4. 迭代 List结合 判断是否为 普通项 如果是普通项 打印 如果是上
传项 保存到磁盘上
                    // false: 上传项
                    // 上传项 有两个主要属性 文件名, IO流
                    String name = fileItem.getName(); // 获取上传的文件名
                    InputStream in = fileItem.getInputStream(); // 获取输入流

```

```

        // 保存到 磁盘上（硬盘）
        // 文件上传的目录    如何获取到当前程序的目录    保存文件需要一个绝对路径
        D:/itheima/xxxx

        String realpath =
req.getContext().getRealPath("upload/"); // 获取当前项目的绝对路径
        System.out.println(realpath);

        // 判断目录是否存在
        File folder = new File(realpath);
        if (folder.exists() == false) { // 判断目录是否存在
            folder.mkdirs(); // 创建目录
        }
        // 创建一个输出流
        FileOutputStream out = new FileOutputStream(realpath +
name); // 参数：绝对路径的磁盘地址
        // 将输入流输出到写成文件
        IOUtils.copy(in, out);
    }
}
} catch (FileUploadException e) {
    e.printStackTrace();
}
}
}
}

```