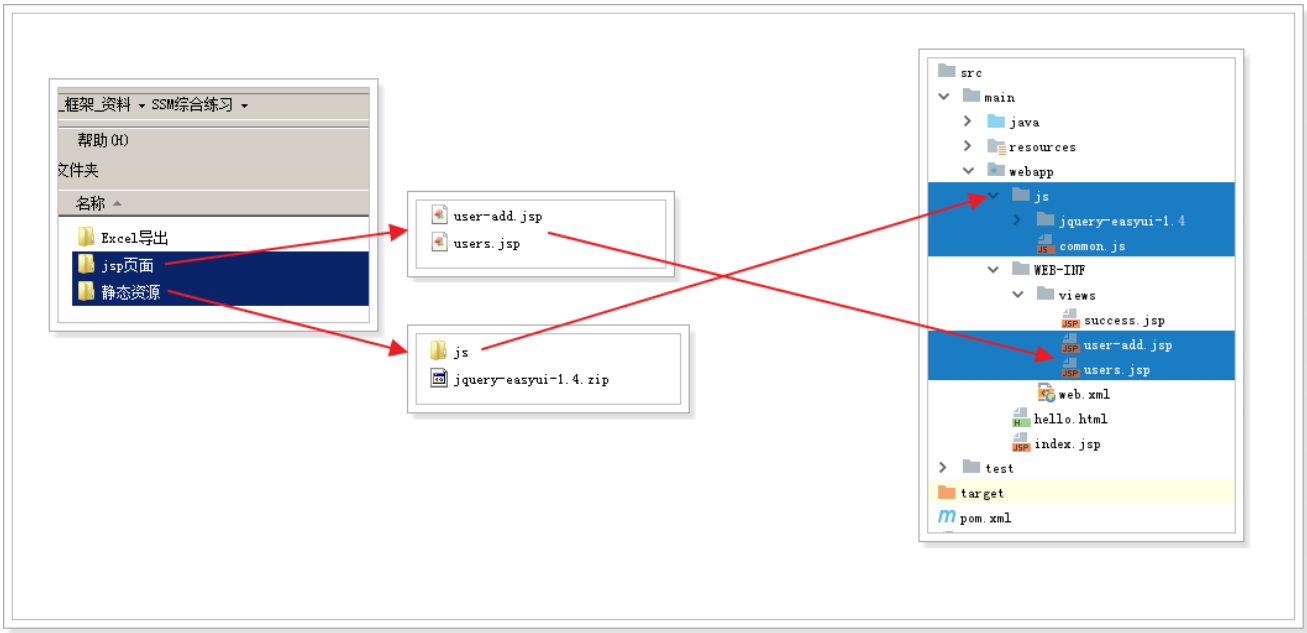
1 准备工作

1.1 导入静态资源和jsp页面

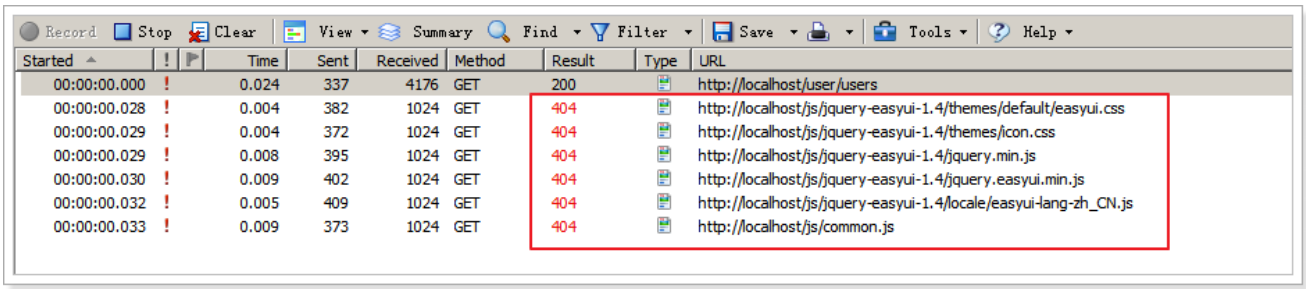


1.2 编写controller,实现简单的页面跳转

```
package cn.itcast.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("user")
public class UserController {

    // 跳转到用户列表页
    @RequestMapping("users")
    public String toUserList() {
        return "users";
    }
}
```

1.3 启动tomcat,测试:

发现没有任何CSS样式，打开开发者工具，发现：



所有的静态资源都是404，什么原因呢？

1.4 解决静态资源404问题

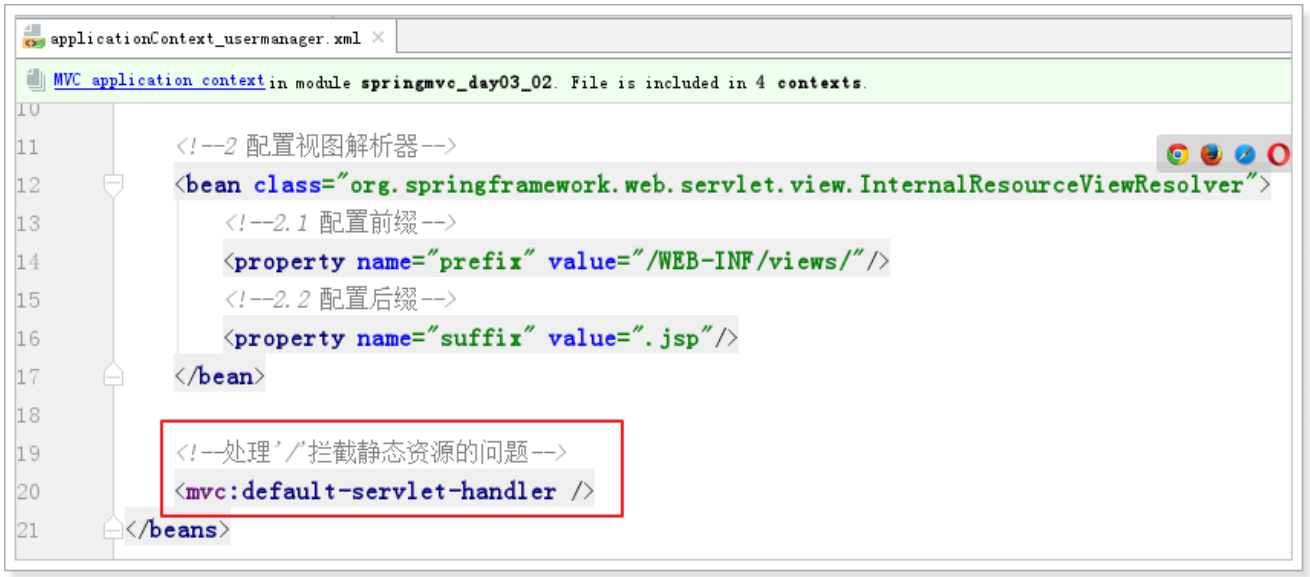
回顾我们的SpringMVC前端控制器的配置：

web.xml

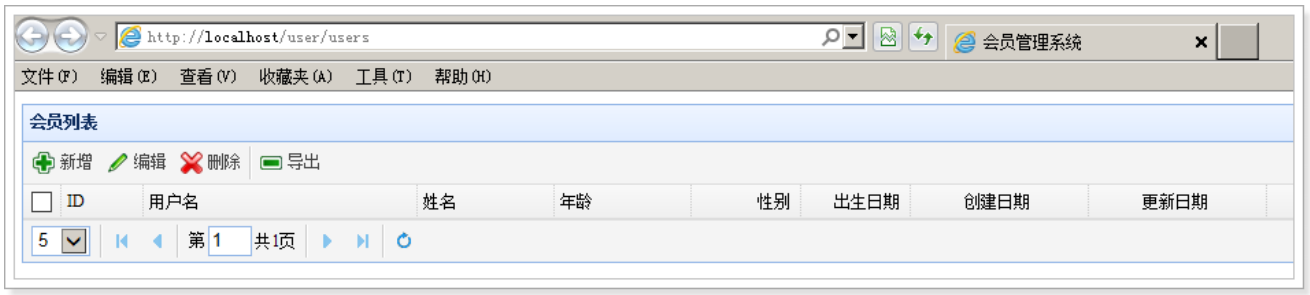
```
17
18      <!--2 整合springmvc-->
19      <servlet>
20          <servlet-name>usermanager</servlet-name>
21          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servl
22          <!--2.1 加载springmvc的配置文件-->
23          <init-param>
24              <param-name>contextConfigLocation</param-name>
25              <param-value>classpath:spring/applicationContext_usermanager.xml</pa
26          </init-param>
27      </servlet>
28      <servlet-mapping>
29          <servlet-name>usermanager</servlet-name>
30          <!--拦截jsp以外的内容-->
31          <url-pattern>/</url-pattern>
32      </servlet-mapping>
33
34  </web-app>
```

由于我们设置DispatcherServlet的映射规则是"/"，所以页面中的所有请求将被拦截，也包括静态资源，后端的Handler是无法处理静态资源的，所有会导致访问静态资源会404，SpringMVC提供了一种解决方案就是加入<mvc:default-servlet-handler/>标签，原理是将静态资源转交给服务器处理。

```
<!-- 处理静态资源被"/"所拦截的问题 -->
<mvc:default-servlet-handler/>
```



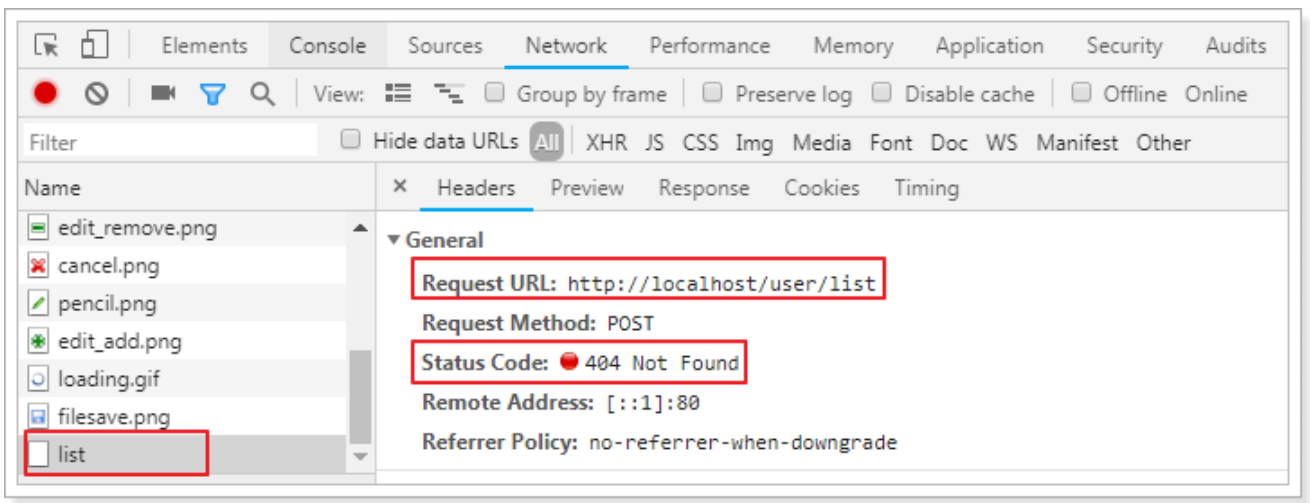
添加配置，再次访问：



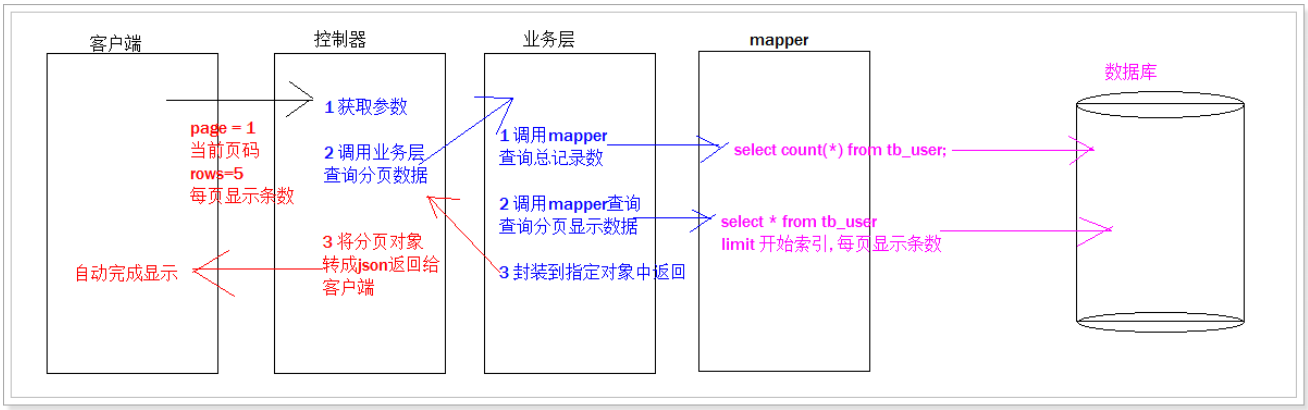
2 用户列表功能

2.1 问题

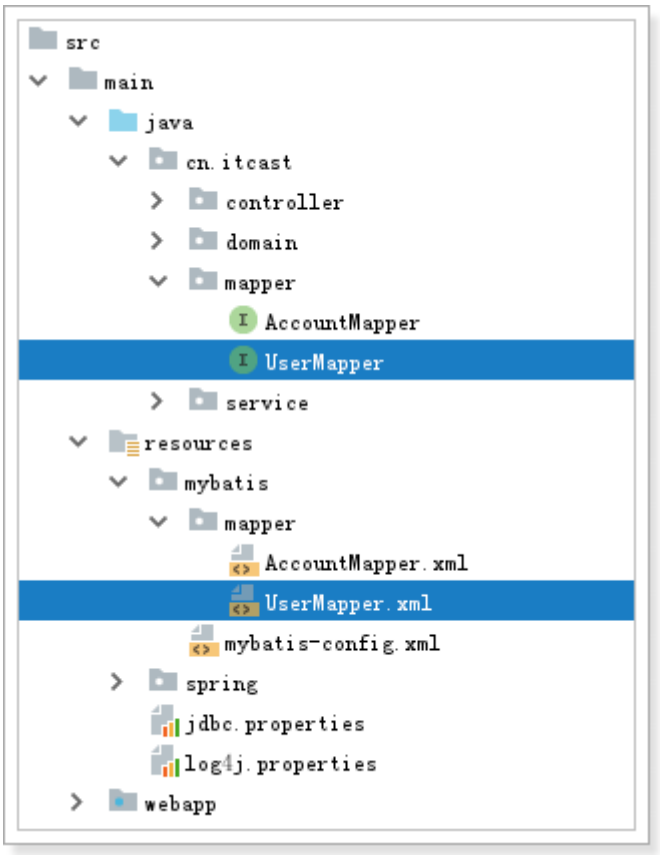
刚才的页面中，目前没有任何数据，原因是获取数据的功能我们还没有写：



解决:



2.2 编写mapper



2.2.1 编写UserMapper.java

```
package cn.itcast.mapper;

import cn.itcast.domain.User;

import org.apache.ibatis.annotations.Param;
```

```
import java.util.List;

public interface UserMapper {
    /**
     * 查询总记录数
     */
    public Long queryTotalCount();

    /**
     * 查询分页显示数据
     */
    List<User> queryUserListByPage(@Param("start") int start,
@Param("rows")int rows);
}
```

2.2.2 编写UserMapper.xml

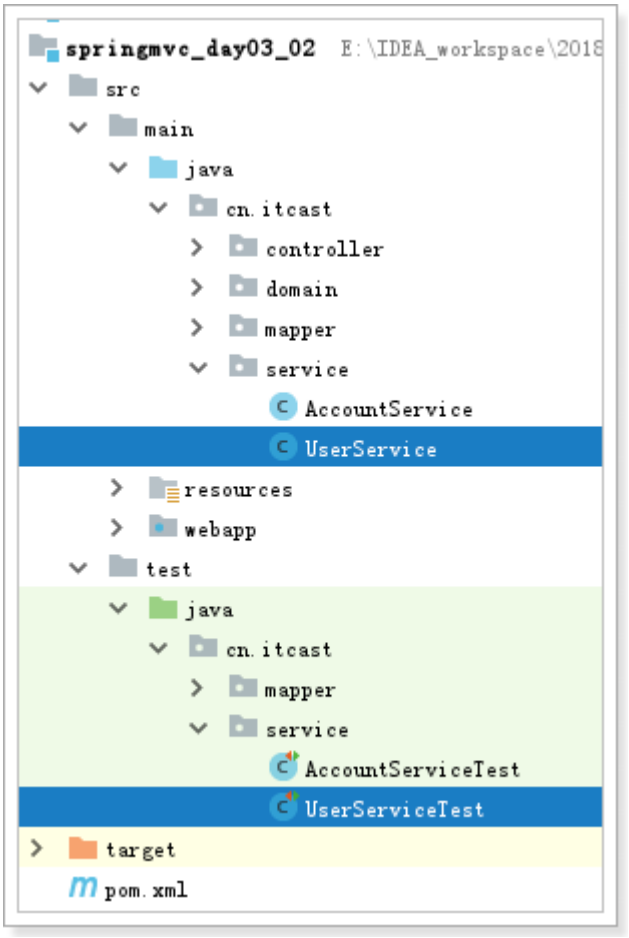
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.itcast.mapper.UserMapper">
    <!--查询总记录数-->
    <select id="queryTotalCount" resultType="Long">
        select count(*) from tb_user
    </select>

    <!--查询分页显示数据-->
    <select id="queryUserListByPage" resultType="User">
        select * from tb_user limit #{start}, #{rows}
    </select>
</mapper>
```

2.2.3 测试

```
package cn.itcast.mapper;

import cn.itcast.domain.User;

import org.junit.Test;
```

```java
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import java.util.List;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:spring/application*.xml")
public class UserMapperTest {

    @Value("#{userMapper}")
    private UserMapper userMapper;

    @Test
    public void queryTotalCount() throws Exception {
        Long totalCount = userMapper.queryTotalCount();
        System.out.println(totalCount);
    }

    @Test
    public void queryUserListByPage() throws Exception {
        List<User> userList = userMapper.queryUserListByPage(2, 2);
        for (User user : userList) {
            System.out.println(user);
        }
    }

}
```

2.3 编写业务层



2.3.1 编写业务类

```
package cn.itcast.service;

import cn.itcast.domain.DataGridResult;
import cn.itcast.domain.User;
import cn.itcast.mapper.UserMapper;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import java.util.List;

@Service("userService")
public class UserService {

    @Value("#{userMapper}")
    private UserMapper userMapper;
    /**
     * 分页查询方法

     * @param page  当前页码
```

```
     * @param rows  每页显示条数
     * @return
     */
    public DataGridResult<User> queryUserListByPage(int page, int rows) {
        // 查询总记录数
        Long total = this.userMapper.queryTotalCount();
        // 查询分页显示数据
        List<User> userList = this.userMapper.queryUserListByPage((page - 1)
* rows, rows);
        // 将数据封装到页面对象中
        DataGridResult<User> dataGridResult = new DataGridResult<User>(total,
userList);
        return dataGridResult;
    }
}
```

2.3.2 测试

```
package cn.itcast.service;

import cn.itcast.domain.DataGridResult;
import cn.itcast.domain.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:spring/application*.xml")
public class UserServiceTest {

    @Value("#{userService}")
    private UserService userService;

    @Test
    public void queryUserListByPage() throws Exception {
        DataGridResult<User> dataGridResult =
userService.queryUserListByPage(1, 3);
        System.out.println("----------------------------------------");
        System.out.println(dataGridResult);
```

```
    }

}
```

2.4 编写 controller

编写一个controller,需要搞清楚三件事:

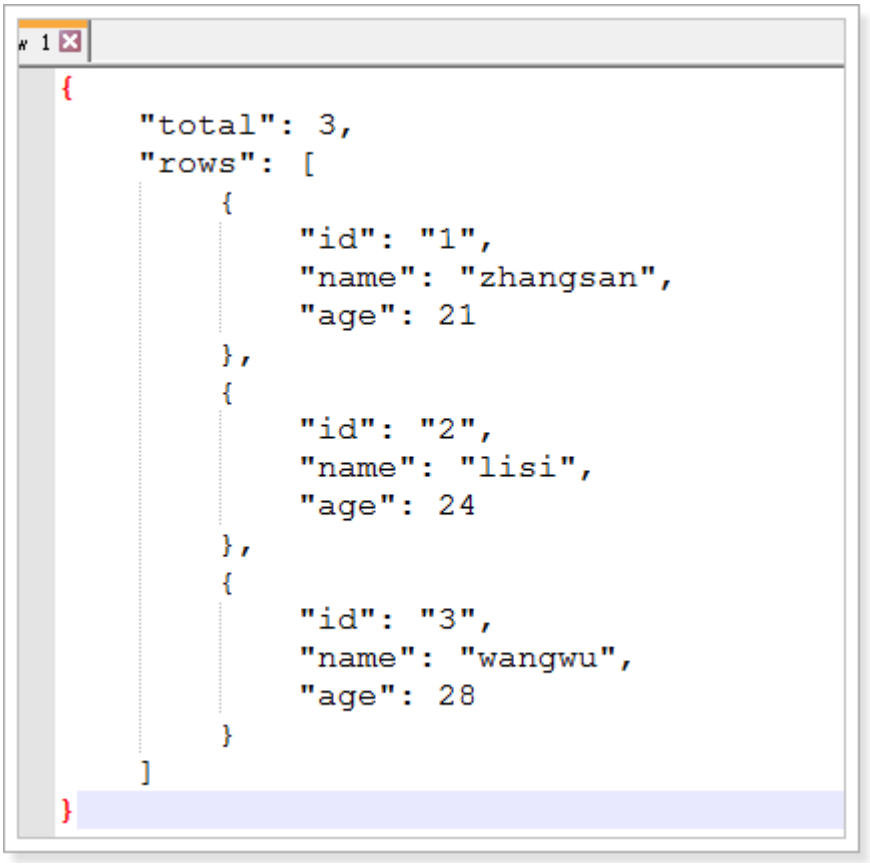
1. 请求的映射路径(RequestMapping映射地址)
2. 响应结果(方法的返回值)
3. 请求参数(方法的参数列表)

2.4.1 请求的映射路径

2.4.2 响应结果(方法的返回值)

返回结果 是通过EasyUI的datagrid来获取数据:

datagird要求的数据格式是：

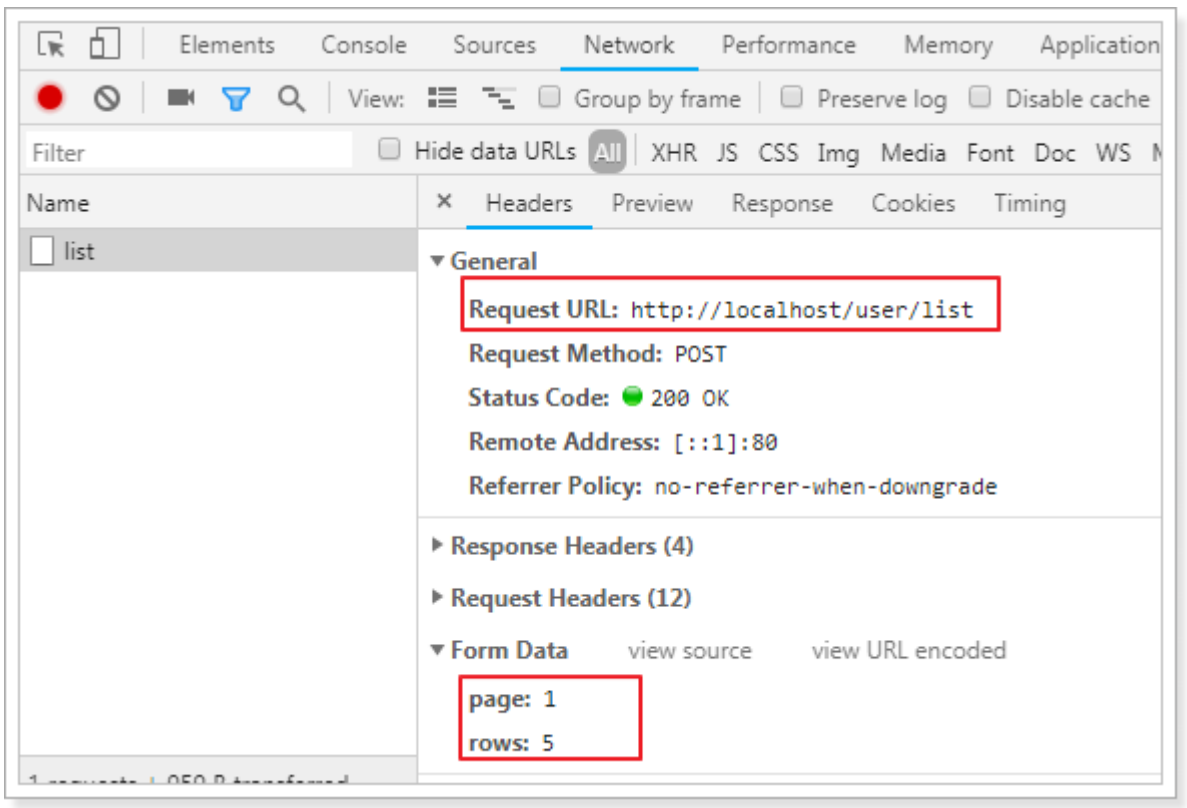


这是一个json对象，包含total和rows属性。total是总条数信息，rows是数据信息，所以我们返回的应该是这样一个对象：

```
package cn.itcast.domain;

import java.util.List;

/**
 * easyui的数据表对象
 */
public class DataGridResult<T> {
    // 总条数
    private long total;

    // 数据
    private List<T> rows;

    public long getTotal() {
        return total;
    }


    public void setTotal(long total) {
```

```
        this.total = total;
    }

    public List<T> getRows() {
        return rows;
    }

    public void setRows(List<T> rows) {
        this.rows = rows;
    }

    @Override
    public String toString() {
        return "DataGridResult{" +
                "total=" + total +
                ", rows=" + rows +
                '}';
    }
}
```

2.4.3 请求参数



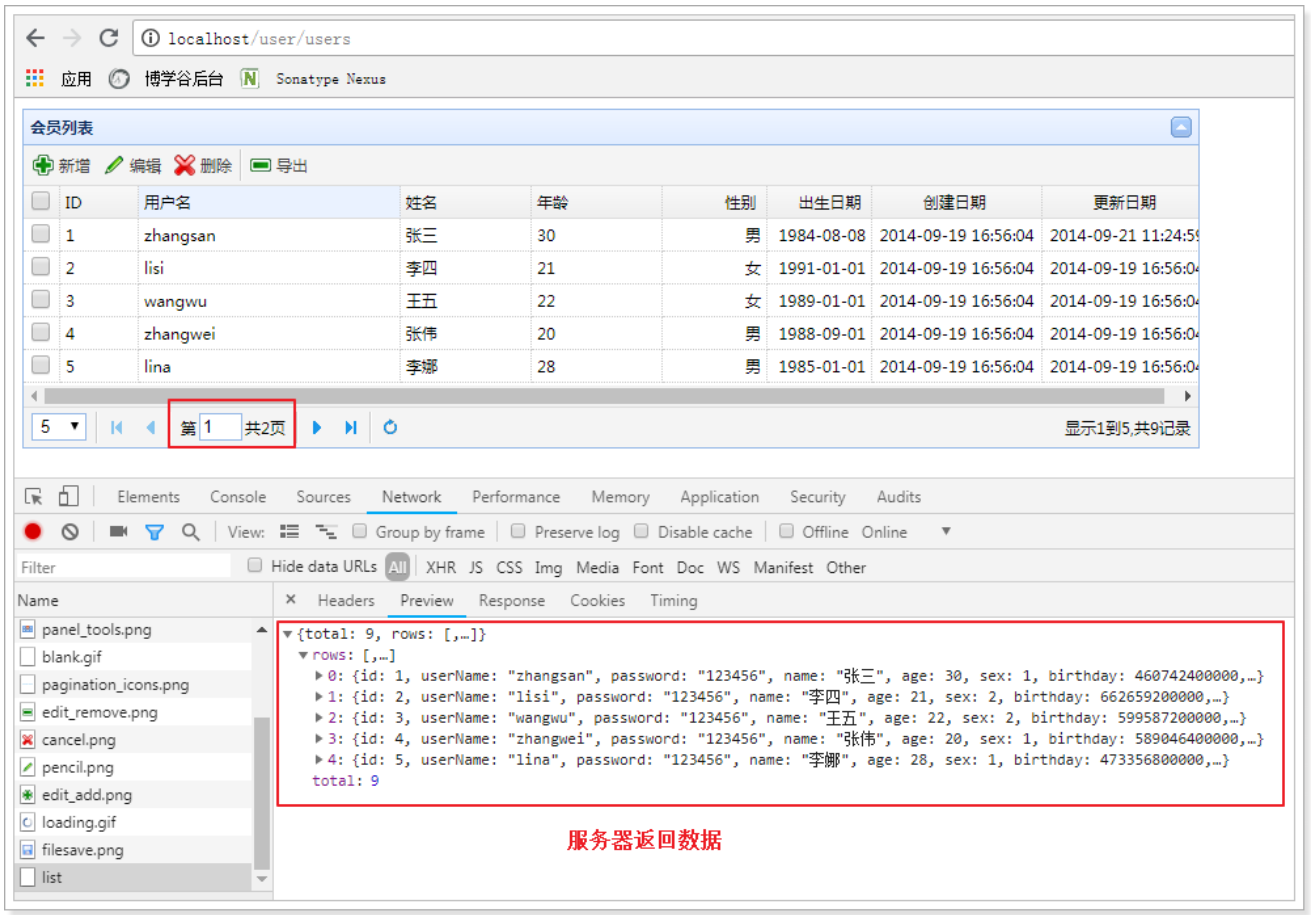
datagrid请求数据的时候，有两个参数：page (当前页数) 和rows(每页条数) 信息。也就是时候需要进行分页查询。我们需要接收这两个参数。

2.4.4 定义controller

```
package cn.itcast.controller;

import cn.itcast.domain.DataGridResult;
import cn.itcast.domain.User;
import cn.itcast.service.UserService;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("user")
public class UserController {

    @Value("#{userService}")
    private UserService userService;

    // 跳转到用户列表页
    @RequestMapping("users")
    public String toUserList() {
        return "users";
    }

    /**
     * 查询分页显示数据
     * @param page  当前页码
     * @param rows  每页显示条数
     * @return
     */
    @RequestMapping("list")
    @ResponseBody
    public DataGridResult<User>
queryUserListByPage(@RequestParam(value="page", defaultValue = "1") int page,
```

```
    @RequestParam(value="rows", defaultValue = "5") int rows){
            return userService.queryUserListByPage(page, rows);
        }
    }
```

2.4.5 测试



3 mybatis分页插件

3.1 原有分页的问题及思考

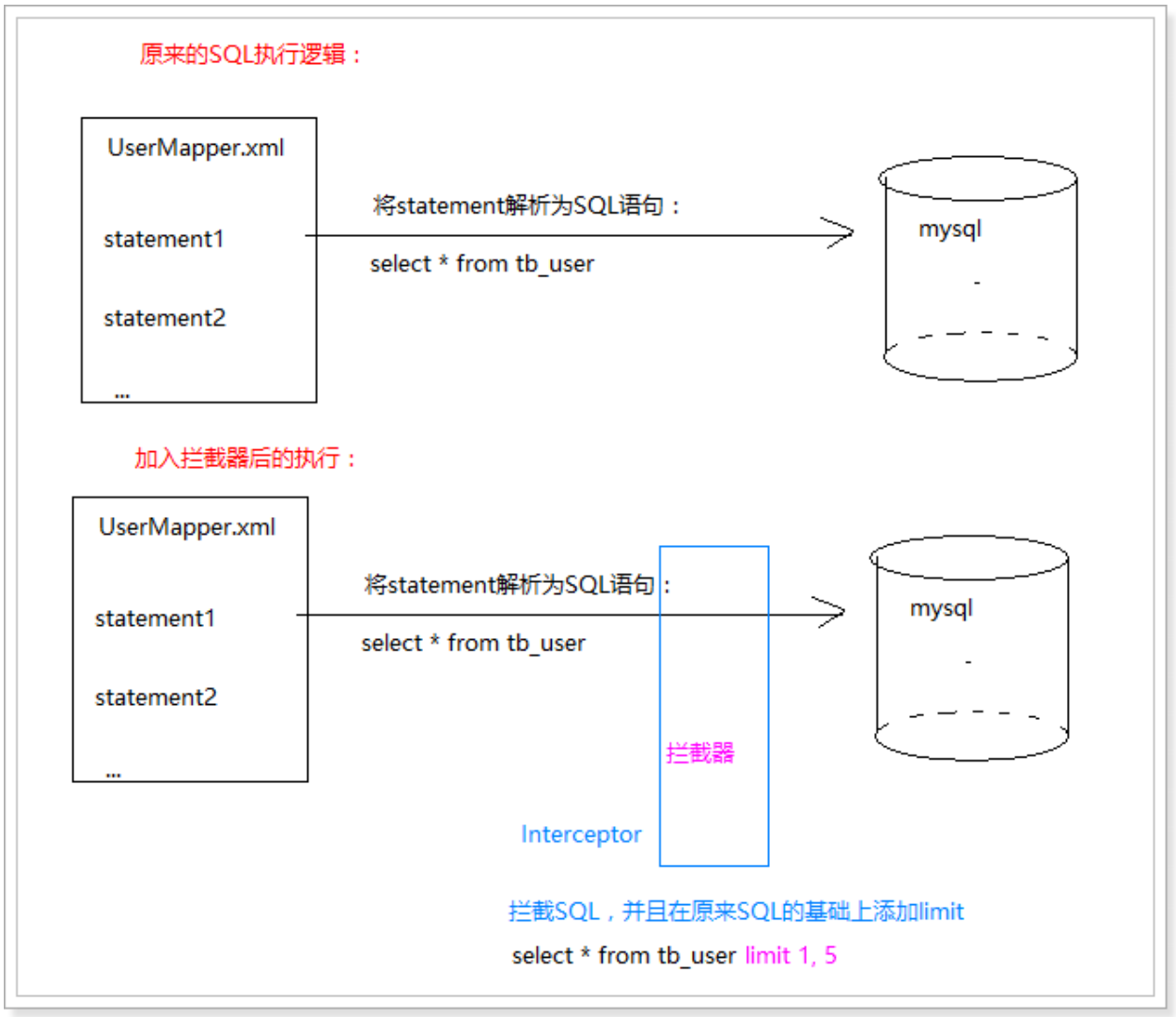
- 思考：我们前面的分页实现存在怎样的问题？
  1）我们需要给每个SQL添加分页的逻辑，还要保留不分页的SQL，重复！
  2）不同数据库分页方式不同，写死后不利于以后的开发和维护！
  3）需要手动查询总条数信息
- 有没有更好的方案呢？

其实分页的逻辑基本是类似的：以MySQL举例，基本都是在原有SQL基础上加上limit start,rows

既然如此，我们完全可以使用mybatis的拦截器机制，在SQL执行前做一个拦截，然后对SQL语句加上limit

这样所有需要分页的SQL就自动实现分页逻辑了！

- 拦截器实现分页的原理图：



3.2 PageHelper分页插件

上面的逻辑已经有大神想到了，并且写出了开源的组件：

网址: https://github.com/pagehelper/Mybatis-PageHelper

MyBatis 分页插件 - PageHelper

English

如果你也在用 MyBatis，建议尝试该分页插件，这一定是**最方便**使用的分页插件。

分页插件支持任何复杂的单表、多表分页，部分特殊情况请看重要提示。

想要使用分页插件？请看如何使用分页插件。

支持 MyBatis 3.1.0+

物理分页

该插件目前支持以下数据库的**物理分页**：

1. `Oracle`
2. `Mysql`
3. `MariaDB`
4. `SQLite`
5. `Hsqldb`
6. `PostgreSQL`
7. `DB2`
8. `SqlServer(2005,2008)`
9. `Informix`
10. `H2`

3.2.1 添加依赖

```
<!-- 分页助手 -->
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>3.7.5</version>
</dependency>
<dependency>
    <groupId>com.github.jsqlparser</groupId>
    <artifactId>jsqlparser</artifactId>
    <version>0.9.1</version>
</dependency>
```

3.2.2 在mybatis-config.xml中配置分页插件：

```
<plugins>
    <!-- 配置分页助手的插件 -->
    <plugin interceptor="com.github.pagehelper.PageHelper">
        <!-- 指定数据库方言 -->
        <property name="dialect" value="mysql"/>
        <!-- 设置为true时，查询结果中会查询出总条数信息 -->
        <property name="rowBoundsWithCount" value="true"/>
    </plugin>
</plugins>
```

3.2.3 添加新的mapper接口

```
package cn.itcast.mapper;

import cn.itcast.domain.User;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface UserMapper {

    /**
     * 查询所有用户
     */
    public List<User> queryUserList();

}
```

3.2.4 编写mapper.xml文件

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="cn.itcast.mapper.UserMapper">
    <!--查询全部用户-->
    <select id="queryUserList" resultType="User">
        select * from tb_user
    </select>
</mapper>
```

3.2.5 修改业务类

```java
package cn.itcast.service;

import cn.itcast.domain.DataGridResult;
import cn.itcast.domain.User;
import cn.itcast.mapper.UserMapper;
import com.github.pagehelper.PageHelper;
import com.github.pagehelper.PageInfo;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import java.util.List;

@Service("userService")
public class UserService {

    @Value("#{userMapper}")
    private UserMapper userMapper;

    /**
     * 分页查询方法
     * @param page  当前页码
     * @param rows  每页显示条数
     * @return
     */
    public DataGridResult<User> queryUserListByPage(int page, int rows) {
        /*

        // 方式一 手动准备分页数据
```

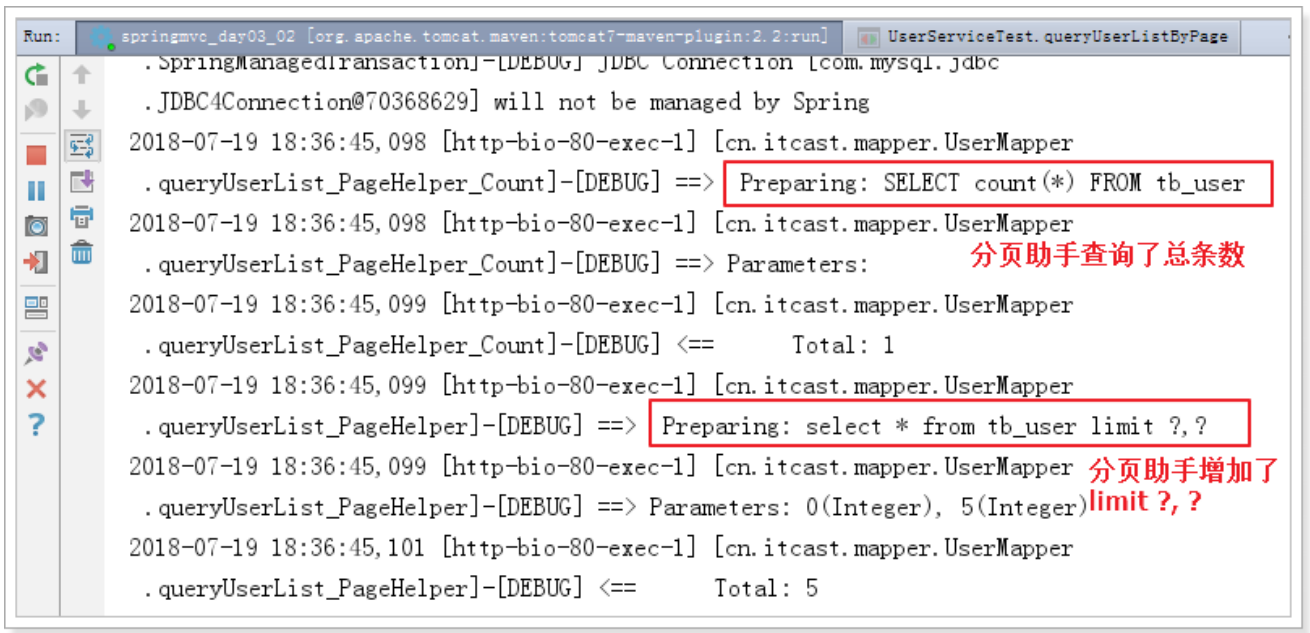
```
        // 查询总记录数
        Long total = this.userMapper.queryTotalCount();
        // 查询分页显示数据
        List<User> userList = this.userMapper.queryUserListByPage((page - 1)
* rows, rows);
        */

        // 方式二：使用分页助手,开始分页，指定两个参数：当前页码，每页条数
        PageHelper.startPage(page, rows);
        // 然后分页拦截器会自动对接下来的查询进行分页
        List<User> userList = userMapper.queryUserList();
        // 封装分页信息对象
        PageInfo<User> pageInfo = new PageInfo<User>(userList);

        // 将数据封装到页面对象中
        DataGridResult<User> dataGridResult = new DataGridResult<User>
(pageInfo.getTotal(), userList);
        return dataGridResult;
    }
}
```

3.2.6 总结

PageHelper做了两件事情：

1）自动查询了总条数信息

2）自动的给我们SQL语句后面添加分页逻辑：limit ?,?



4 新增用户

4.1 通用的页面跳转

- 查看新增用户按钮

```
<div id="userAdd" class="easyui-window" title="新增会员"
    data-options="modal:true,closed:true,iconCls:'icon-save',href:'/user/user-add'"
    style="width:800px;height:600px;padding:10px;">
        The window content.
</div>
```

需要跳转页面到user-add.jsp

- 我们写了两个页面跳转的逻辑：

```
/**
 * 页面跳转功能，进入用户列表页面
 * @return
 */
@RequestMapping("/users")
public String toUserList(){
    return "users";
}
/**
 * 页面跳转功能，进入添加用户页面
 * @return
 */
@RequestMapping("/user-add")
public String toUserAdd(){
    return "user-add";
}
```

可以发现，页面跳转的逻辑很简单，并且有很大的共性，就是用户输入的路径，与用户要跳转的页面一般是一样的！

既然如此，我们是否可以考虑把RequestMapping的映射路径中的值，作为返回的视图名称呢？

这时我们就可以想到一个东西：@PathVariable

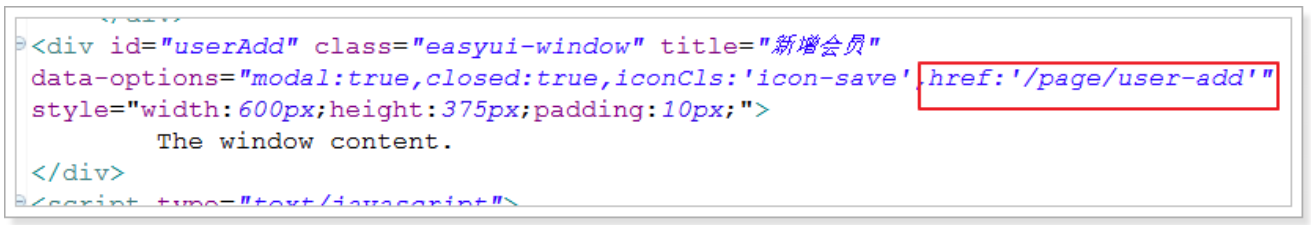
- 通用的页面跳转：

```
@Controller
@RequestMapping("page")
public class PageController {

    @RequestMapping("{pageName}")
    // 通过PathVariable获取用户输入的路径，把路径直接作为视图名称返回。用户输入的必须是正确的视图名称
    public String toPage(@PathVariable("pageName") String pageName){
        return pageName;
    }
}
```

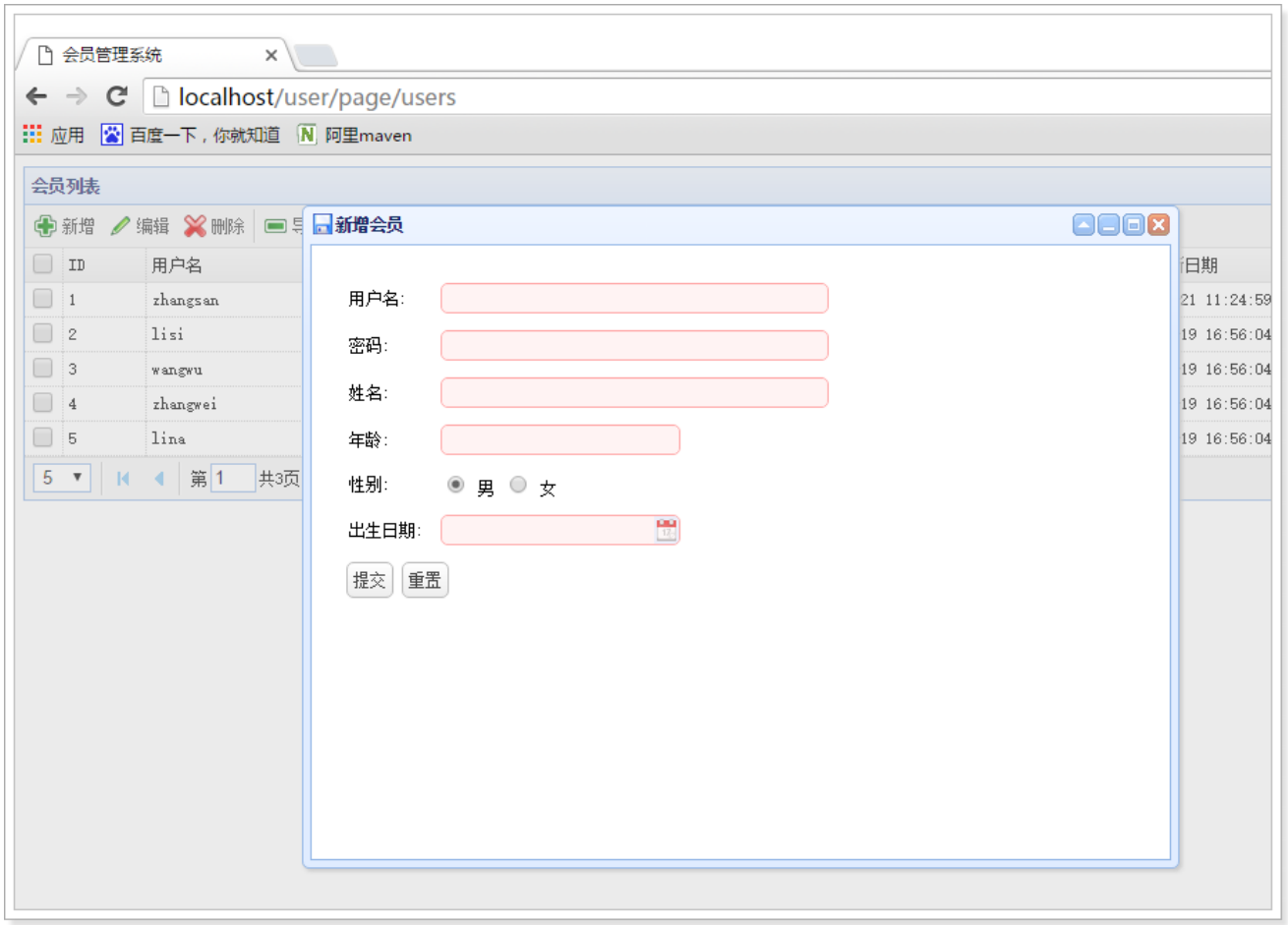
这样，我们访问首页就应该输入：http://localhost/page/users

最后的users 就被赋值给了 pageName ，然后作为视图名称，就能找到 WEB-INF/views/users.jsp了

然后新增用户的页面跳转也要改变：http://localhost/page/user-add



- 访问测试



4.2 编写Controller

分析form表单的提交逻辑：

```
<script type="text/javascript">
    function submitForm(){
        if(!$('#content').form('validate')){
            $.messager.alert('提示','表单还未填写完成!');
            return ;
        }
        $.post("/user/save",$("#content").serialize(), function(data){
            if(data.status == 200){
                $.messager.alert('提示','新增会员成功!');
                $('#userAdd').window('close');
                $("#userList").datagrid("reload");
                clearForm();
            }else{
                $.messager.alert('提示','新增会员失败!');
            }
        });
    }
    function clearForm(){
        $('#content').form('reset');
    }
</script>
```

请求路径

返回的数据结果中必须有status属性

返回值：需要包含一个status属性，这里可以考虑用一个Map<String,Integer>，我们也可以自定义对象

```
/**
 * 返回的通用页面结果
 */
public class PageResult {
    private Integer status;// 结果的状态码
    private Object data;// 结果的数据

    public PageResult() {
    }

    public PageResult(Integer status) {
        super();
        this.status = status;
    }

    public PageResult(Integer status, Object data) {
        super();
        this.status = status;
        this.data = data;
    }

    public Integer getStatus() {
        return status;
    }
    // 提供一个默认的成功状态方法
    public static PageResult ok(){
        return new PageResult(200);
    }
    // 提供一个默认的异常状态方法
    public static PageResult error(){
        return new PageResult(500);
    }

    public void setStatus(Integer status) {
        this.status = status;
    }


    public Object getData() {
```

```
        return data;
    }

    public void setData(Object data) {
        this.data = data;
    }
}
```

参数：用户的所有信息，这里用User对象>

```
@RequestMapping("save")
@ResponseBody
public PageResult saveUser(User user){
    try {
        this.userService.saveUser(user);
        // 返回成功状态200
        return PageResult.ok();
    } catch (Exception e) {
        e.printStackTrace();
        // 返回异常状态500
        return PageResult.error();
    }
}
```

4.3 编写UserService

```
/**
 * 新增用户的功能:
 * @param user
 */
public void saveUser(User user) {
    this.userMapper.insertUser(user);
}
```

4.4 编写UserMapper

```
/**
 * 新增用户
 * @param user
 */
void insertUser(User user);
```

4.5 编写sql

```
    <insert id="insertUser" parameterType="User" useGeneratedKeys="true"
keyProperty="id" keyColumn="id">
        INSERT INTO tb_user (
            id,
            user_name,
            password,
            name,
            age,
            sex,
            birthday,
            created,
            updated
        )
        VALUES
        (
            NULL,
            #{userName},
            #{password},
            #{name},
            #{age},
            #{sex},
            #{birthday},
            NOW(),
            NOW()
        );
    </insert>
```

4.6 测试: 解决日期转换问题

提交请求后报错：

```
nework.validation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1 errors
t message [Failed to convert property value of type 'java.lang.String' to required type 'java.util.Date' for property 'birthday
lidation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1 errors
t message [Failed to convert property value of type 'java.lang.String' to required type 'java.util.Date' for property 'birthday
ation.BindException: org.springframework.validation.BeanPropertyBindingResult: 1 errors
t message [Failed to convert property value of type 'java.lang.String' to required type 'java.util.Date' for property 'birthday
```

查看后发现是因为类型转换问题，我们在页面中的birthday字段获取的是字符串类型，而User中的是Date类型，怎么办？

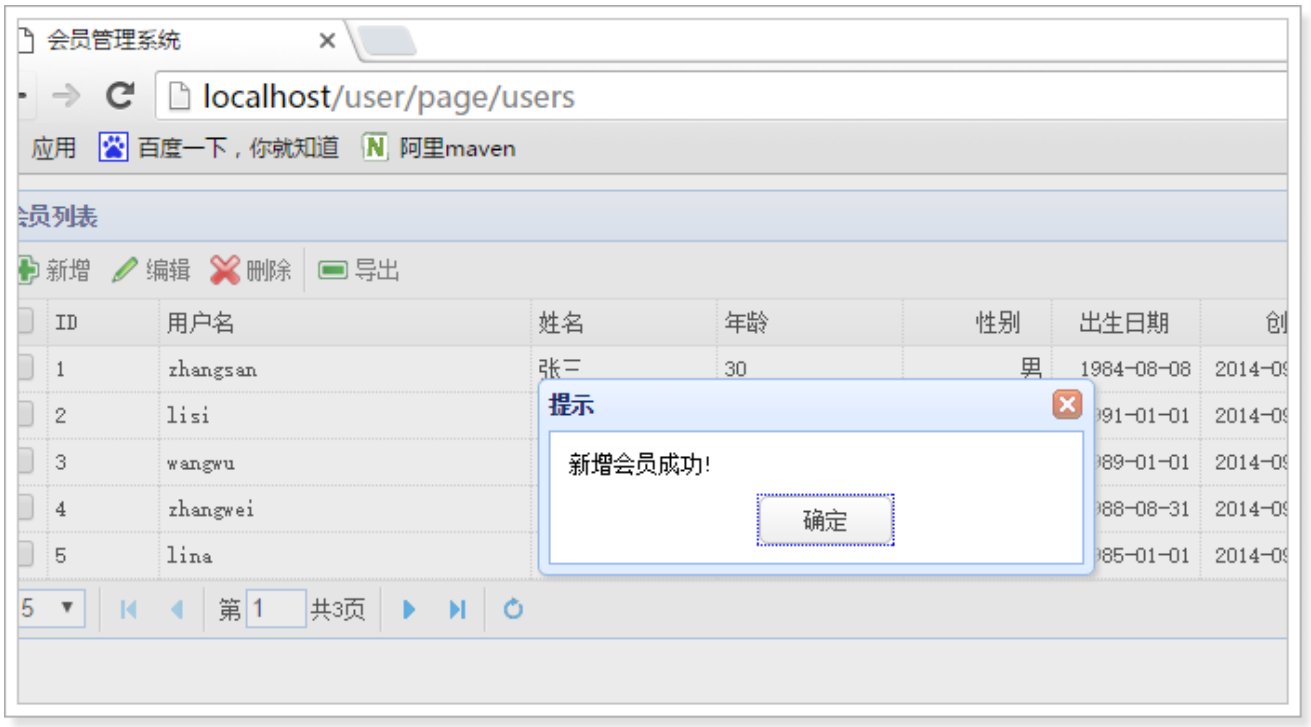
SpringMVC提供了字符串转日期类型的机制，使用@DateTimeFormat注解即可解决。

```
@DateTimeFormat(pattern="yyyy-MM-dd")
private Date birthday;
```

修改User类

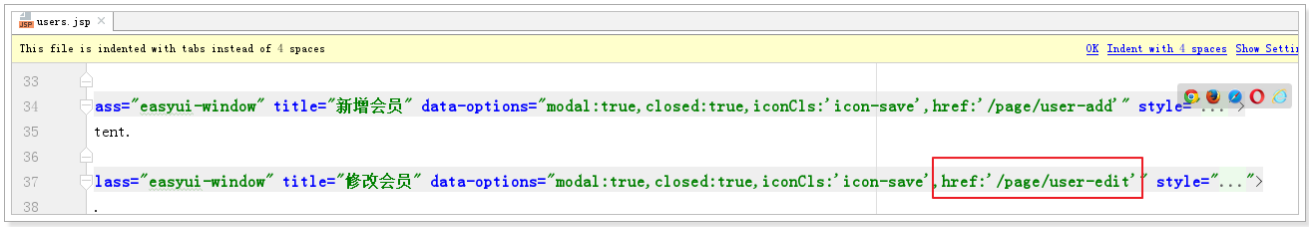
```
// 出生日期
@DateTimeFormat(pattern="yyyy-MM-dd")
private Date birthday;
```

再次测试:



5 修改用户(课后作业)

5.1 通用页面跳转



5.2 编写controller

```
/**
 * 修改用户
 * @param user
 * @return
 */
@RequestMapping("edit")
@ResponseBody
public PageResult editUser(User user) {
    try {
        userService.editUser(user);
        return PageResult.ok();
    } catch(Exception e) {
        e.printStackTrace();
        return PageResult.error();
    }
}
```

5.3 编写 UserService

```
/**
 * 修改用户
 */
public void editUser(User user) {
    userMapper.editUser(user);
}
```

5.4 编写 UserMapper

```java
/**
 * 修改用户
 */
public void editUser(User user);
```

5.5 编写sql

```xml
<!--修改用户-->
<update id="editUser">
    update tb_user
    set
        user_name=#{userName},
        password=#{password},
        name=#{name},
        age=#{age},
        sex=#{sex},
        birthday=#{birthday},
        updated=now()
    where id=#{id}
</update>
```

测试即可!

6 删除用户

6.1 编写Controller

- **查看删除的请求格式**

```
text:'删除',
iconCls:'icon-cancel',
handler:function(){
    var ids = getSelectionsIds();
    if(ids.length == 0){
        $.messager.alert('提示','未选中用户!');
        return ;
    }
    $.messager.confirm('确认','确定删除ID为 '+ids+' 的会员吗? ',function(r){
        if (r){
            $.post("/user/delete",{'ids':ids}, function(data){
                if(data.status == 200){
                    $.messager.alert('提示','删除会员成功!',undefined,function(){
                        $("#userList").datagrid("reload");
                    });
                }
            });
        }
    });
}
```

1）返回值：依然要求有一个status参数，所以可以定义一个Map<String,Integer>

2）参数：这里是ids，其实就是多个ID以”,”隔开的字符串。对应到Java中就是数组,或者集合

```
> getSelectionsIds()
<- "8,9"
```

所以，我们的Controller这样定义：

```
/**
 * 新增用户
 * @param user
 * @return
 */
@RequestMapping(value="delete",method=RequestMethod.POST)
@ResponseBody
public Map<String,Integer> deleteUser(@RequestParam("ids")Long[] ids){
    Map<String,Integer> result = new HashMap<>();
    try {
        // 调用Service的删除功能
        this.userService.deleteUserByIds(ids);
        result.put("status", 200);
    } catch (Exception e) {
        e.printStackTrace();
        result.put("status", 500);
    }
    return result;
}
```

6.2 编写UserService

```java
/**
 * 删除用户功能
 * @param ids
 */
public void deleteUserByIds(Long[] ids) {
    this.userMapper.deleteUserByIds(ids);
}
```

6.3 编写UserMapper

```java
/**
 * 删除用户
 * @param ids
 */
void deleteUserByIds(@Param("ids")Long[] ids);
```

6.4 编写mapper.xml

```xml
<delete id="deleteUserByIds">
    DELETE FROM tb_user WHERE id IN
    <foreach collection="ids" item="id" separator="," open="(" close=")">
        #{id}
    </foreach>
</delete>
```

6.5 测试

```
hronization for SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSessior
nnection [com.mysql.jdbc.JDBC4Connection@40e41ed7] will be managed by Spring
paring: DELETE FROM tb_user WHERE id IN ( ? , ? )
meters: 6(Long), 7(Long)
pdates: 2
ession [org.apache.ibatis.session.defaults.DefaultSqlSession@270e0506]
committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@270e
```