基于Hibernate与Spring MVC框架的小型书店管理网站

北京师范大学珠海分校信息技术学院 李明皓

2019年12月19日

01
项目基本介绍

1.1 网站功能

1.2 目录结构

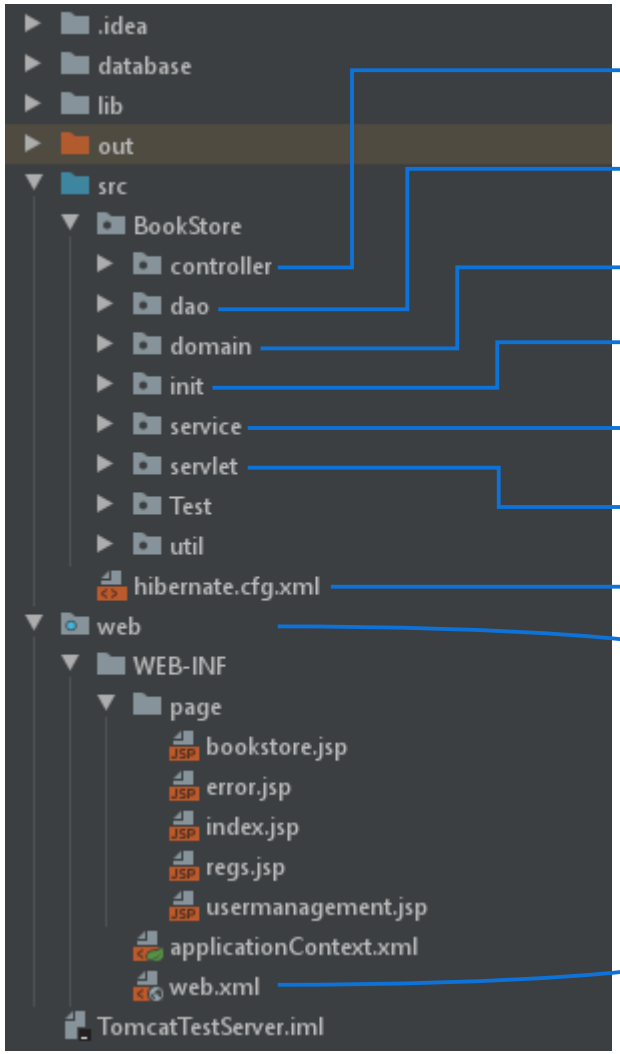
1.3 数据库关系

1.4 框架说明

1.1 网站功能


欢迎来到 BookStore 登录页面
localhost/welcome
欢迎来到 BookStore 登录界面
用户名:
密码:
登录
↓↓↓ 第一次登录网址请注册账号
没有账号?
注册

1.2 目录结构



Spring MVC框架下的C层(controller) 用于处理与用户的交互

Hibernate框架下的DAO层 包含了各种数据库持久化接口

ORM → 实体类映射数据库表

用注解代替的web.xml

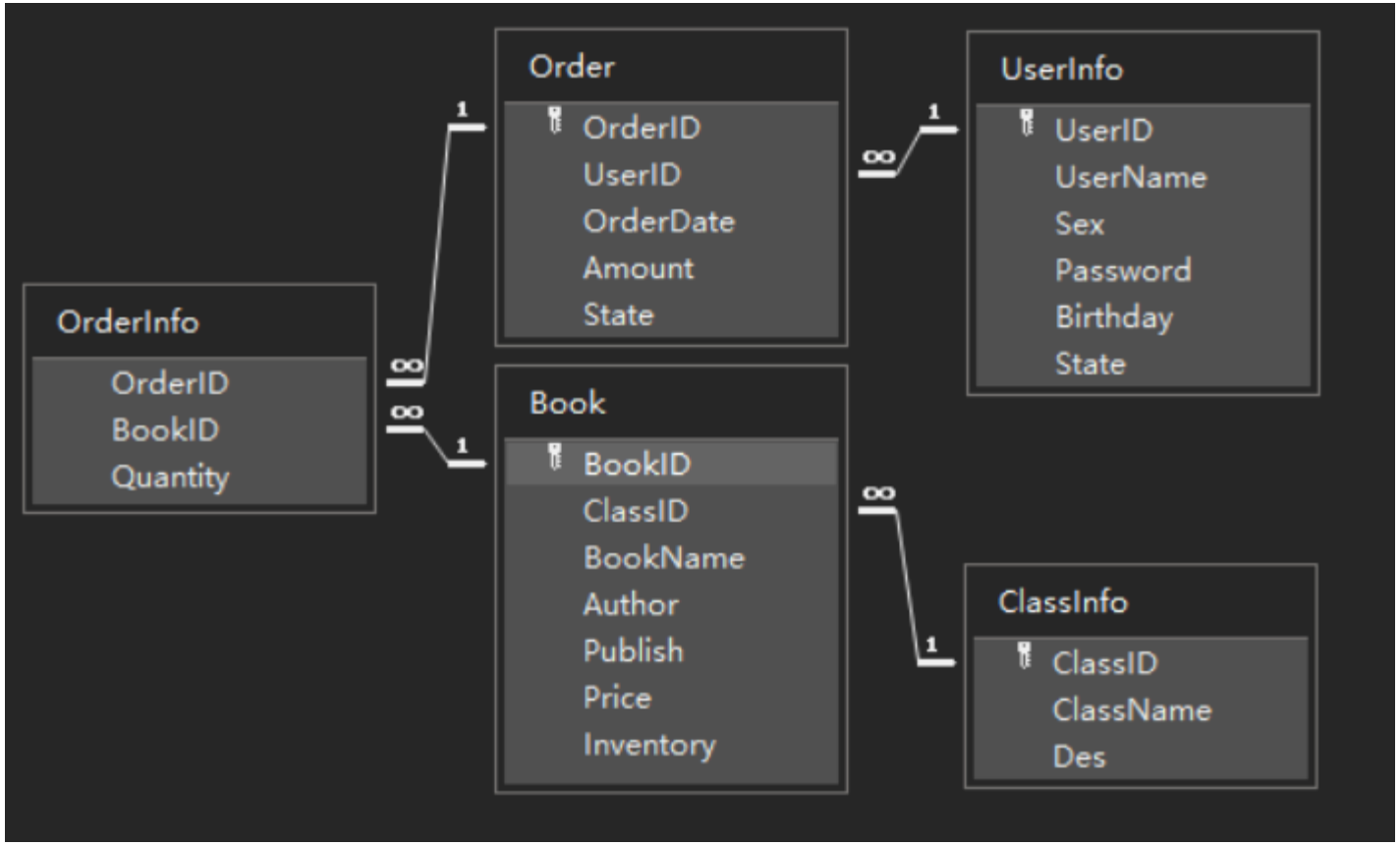
业务逻辑层 使用DAO中提供的接口操作数据库

Servlet 请求处理类

Hibernate的配置文件 用于连接数据库和映射数据库表

Tomcat服务器中部署的网页和配置文件

1.3 数据库关系


Order
OrderID
UserID
OrderDate
Amount
State
UserInfo
UserID
UserName
Sex
Password
Birthday
State
OrderInfo
OrderID
BookID
Quantity
Book
BookID
ClassID
BookName
Author
Publish
Price
Inventory
ClassInfo
ClassID
ClassName
Des
1
∞
1
∞
∞
1
∞
1

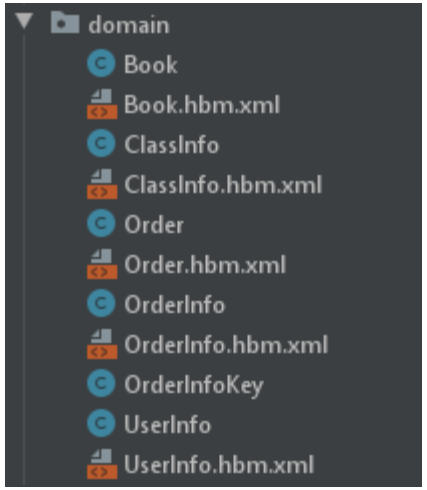
1.4 框架介绍 —— Hibernate的配置

JDBC四大要素

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

    <session-factory>

        <!-- 配置关于数据库连接的四个项: driverClass  url username password -->
        <property name="hibernate.connection.driver_class">net.ucanaccess.jdbc.UcanaccessDriver</property>
        <!--由于服务器运行时hibernate所在的相对目录不相同，所以这里使用了绝对路径-->
        <property name="hibernate.connection.url">jdbc:ucanaccess://F:/TomcatTestServer/database/BookStore.accdb</prope
        <property name="hibernate.connection.username"></property>
        <property name="hibernate.connection.password"></property>

        <!-- 可以将向数据库发送的SQL语句显示出来 -->
        <property name="hibernate.show_sql">false</property>
        <!-- 格式化SQL语句 -->
        <property name="hibernate.format_sql">false</property>

        <!-- hibernate的方言 -->
        <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>

        <!-- 配置Hibernate映射实体类读取其中注解 -->
        <mapping class="BookStore.domain.OrderInfo"/>
        <mapping class="BookStore.domain.Order"/>
        <mapping class="BookStore.domain.UserInfo"/>
        <mapping class="BookStore.domain.Book"/>
        <mapping class="BookStore.domain.ClassInfo"/>
        <mapping class="BookStore.domain.OrderInfoKey"/>
```

配置实体类映射（通过注解）

1.4 框架介绍 —— Hibernate的实现ORM



实体类两种方法
注解和XML

```java
package BookStore.domain;

import javax.persistence.Table;
import javax.persistence.*;

@Entity
@Table(name = "Book")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int BookID;

    private int ClassID;
    private String BookName;
    private String Author;
    private String Publish;
    private float Price;
    private int Inventory;

    public Book() {
    }

    public Book(Book b) {
        this.Author = b.Author;
        this.BookID = b.BookID;
        this.BookName = b.BookName;
        this.ClassID = b.ClassID;
        this.Inventory = b.Inventory;
        this.Price = b.Price;
        this.Publish = b.Publish;
    }

    public Book(int ID) { this.BookID = ID; }

    @Override
    public String toString() {
        return ("\tBookName: " + BookName + "\n\tBookID: "
    }
```

注解方法

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="BookStore.domain">
    <!—
        name: 即实体类的全名
        table: 映射到数据库里面的那个表的名称
        catalog: 数据库的名称
     —>
    <class name="UserInfo" table="UserInfo">
        <!— class下必须要有一个id的子元素 —>
        <!— id是用于描述主键的 —>
        <id name="UserID" column="UserID">
            <!— 主键生成策略 —>
            <generator class="native"></generator>
        </id>
        <!—
            使用property来描述属性与字段的对应关系
            如果length忽略不写，且你的表是自动创建这种方案，那么length的默认长度是255

            <property name="" column=""></property>

        —>
        <property name="Sex" column="Sex"></property>
        <property name="Password" column="Password"></property>
        <property name="Birthday" column="Birthday"></property>
        <property name="State" column="State"></property>
    </class>
</hibernate-mapping>
```

XML方法

1.4 框架介绍 —— Hibernate的联合主键

```java
package BookStore.domain;

import javax.persistence.*;
import java.io.Serializable;

@Entity
@Table(name = "OrderInfo")
@IdClass(OrderInfoKey.class)
//Hibernate操作无主键的OrderInfo表需要联合主键
public class OrderInfo implements Serializable {

    //联合主键开始

    @Id
    @Column(name = "BookID")
    private int BookID;

    @Id
    @Column(name = "OrderID")
    private int OrderID;

    //联合主键结束

    @Column(name = "Quantity", nullable = false)
    private int Quantity;

    public int getBookID() { return BookID; }

    public void setBookID(int bookID) { BookID = bookID; }

    public int getOrderID() { return OrderID; }

    public void setOrderID(int orderID) { OrderID = orderID; }

    public int getQuantity() { return Quantity; }

    public void setQuantity(int quantity) { Quantity = quantity; }
}
```

操作无主键的表时用到的联合主键

```java
package BookStore.domain;

import java.io.Serializable;

public class OrderInfoKey implements Serializable {
    private static final long serialVersionUID = 3176972128965536016L;
    private int OrderID;
    private int BookID;

    public OrderInfoKey() {
    }

    public OrderInfoKey(int OrderID, int BookID) {
        this.OrderID = OrderID;
        this.BookID = BookID;
    }
}
```

联合主键类

配置好的联合主键显示在idea中

1.4 框架介绍 —— Hibernate实现DAO层

```java
package BookStore.dao;

import org.hibernate.query.Query;

//数据库访问（持久化）接口
public interface BookStoreDao {

    //从domain包中获取实体类生成映射文件到数据库表
    void makeConnection();

    //作为用户登录购书；成功登录返回UserID，用户名或密码错误返回值 -1
    int login(String name, String password);

    //增：  在数据库中插入记录并返回主键ID
    int insert(Object o);

    //增：  仅在数据库中插入记录（用于没有主键的表的插入）
    void insertWithoutReturn(Object o);

    //删：  从数据库中删除某条记录
    void delete(Object o);

    //改：  从数据库中更新某条记录
    void update(Object o);

    //查：  从HQL语句获得Query（可以获得类似于JDBC的ResultSet）
    Query getQueryByHQL(String HQL);

    //查：  从SQL语句获得Query
    Query getQueryBySQL(String SQL);

}
```

DAO层中提供的各种接口
实现最基本的增删改查功能

regs.jsp | UserServiceImpl.java | BookStoreService.java | UserService.java | BookStoreServiceImpl.java | bookstc

```java
//用户服务 业务逻辑层接口
public interface UserService {

    //注册新用户
    void register(String username, String password, String sex, Date
     birthday, String state);

    //更改密码
    void changePassword(String newPassword);

    //注销用户
    void deleteUser();

}
```

BookStoreService.java

```java
//书店服务 业务逻辑层接口
public interface BookStoreService {

    //打印书单
    String printBooks();

    //输入 BookID 和购买数量买书
    void buyBook(int bookID, int num);

    //打印订单
    void printOrderInfo();

    //取消正在"Shipping"状态下的订单
    void cancelOrder(int OrderID);
```

使用DAO层接口实现用户服务和购书服务的
业务逻辑层

1.4 框架介绍 —— Spring MVC

```java
package BookStore.init;


import BookStore.servlet.ServletConfig;
import org.springframework.web.servlet.support
 .AbstractAnnotationConfigDispatcherServletInitializer;

//使用注解代替原来的web.xml文件
public class WebInitializer extends
 AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() { return new Class[0]; }

    @Override
    protected Class<?>[] getServletConfigClasses() { return new
     Class[]{ServletConfig.class}; }

    //截获所有请求交给 BookStoreWebConfig.class 处理
    @Override
    protected String[] getServletMappings() { return new String[] {"/"}; }
}
```

交给ServletConfig类去处理请求

截获URL中的 / 符

1.4 框架介绍 —— Spring MVC

```java
package BookStore.servlet;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

//Servlet 配置类
@Configuration
@ComponentScan({"BookStore.controller"})
public class ServletConfig {
    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver vr = new InternalResourceViewResolver();
        vr.setPrefix("/WEB-INF/page/");
        vr.setSuffix(".jsp");
        return vr;
    }
}
```

在controller包中扫描处理这个请求的控制器

配置网页路径的前缀和后缀

1.4 框架介绍 —— Spring MVC

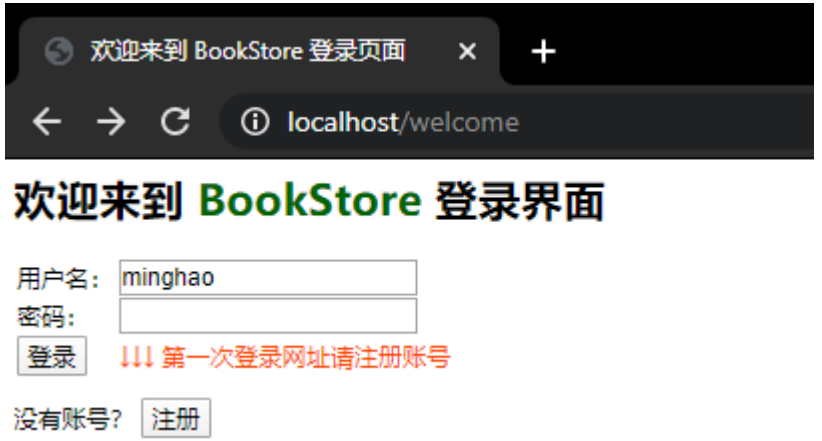
```java
@Controller
@RequestMapping("/bookstore")
public class BookStoreController {

    @RequestMapping("/showBooks")
    public ModelAndView showBooks(HttpSession session,
                                  HttpServletRequest request,
                                  HttpServletResponse response) {
        ModelAndView mv = new ModelAndView();
        try {
            BookStoreServiceImpl bssi = (BookStoreServiceImpl) session.getAttribute(s: "bssi");
            List<Book> bookList = bssi.getBookList();
            System.out.println(bookList);
            String jsonString = JSONObject.toJSONString(bookList);
            mv.setViewName("bookstore");
            mv.addObject(attributeName: "json", jsonString);
            session.setAttribute(s: "bssi", bssi);
            return mv;
        } catch (Exception e) {
            mv.setViewName("error");
            mv.addObject(attributeName: "message", e.toString());
            return mv;
        }
    }

    @RequestMapping("/buyBooks")
    public ModelAndView buyBooks(HttpSession session,
                                 @RequestAttribute("bookID") int bookID,
                                 @RequestAttribute("qty") int qty,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
        ModelAndView mv = new ModelAndView();
```

使用注解标识该类为SpringMVC框架中的一个控制器

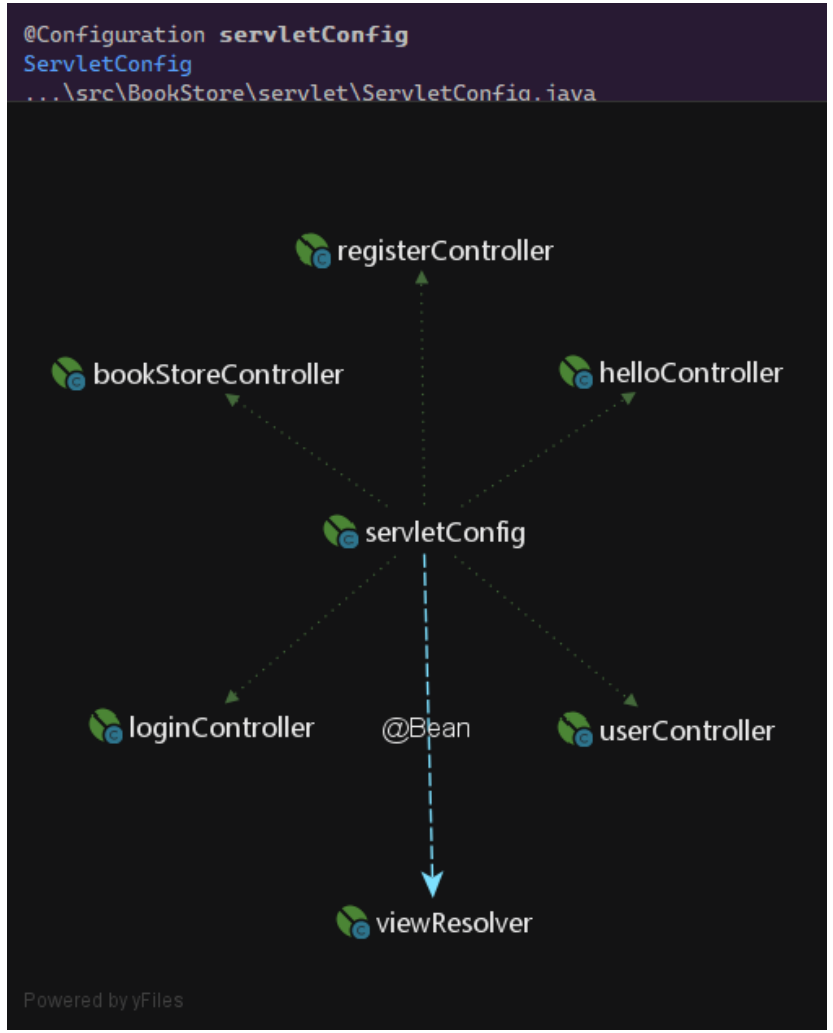
完成"请求路径"到"处理该请求方法"之间的映射关系

e.g.: 使用showBooks控制器的访问路径就是：

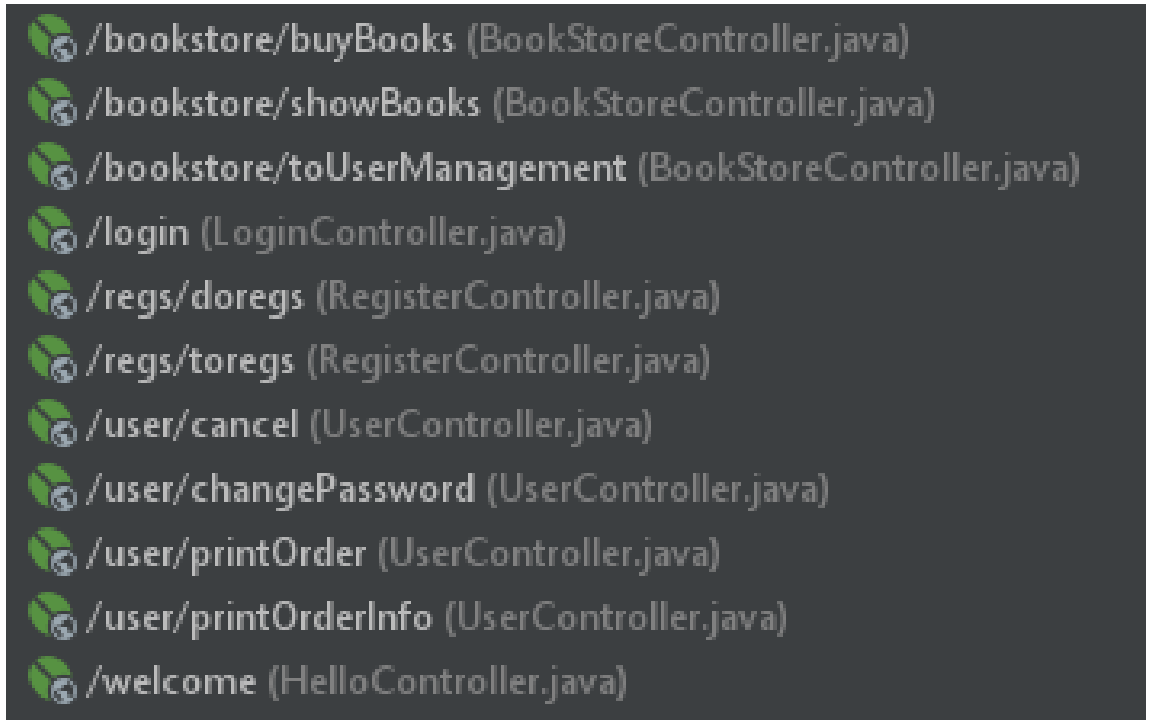
localhost/bookstore/showbooks



1.4 框架介绍 —— Spring MVC



Servlet控制的多个Controller



网站所有的RequestMapping

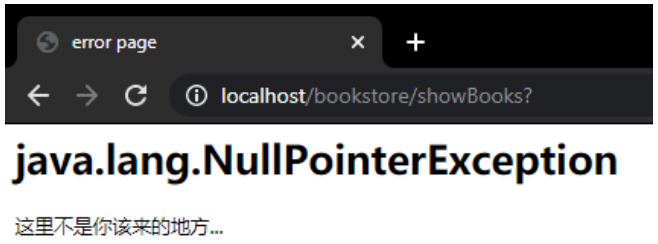
1.4 框架介绍 —— Spring MVC

```java
@Controller
public class LoginController {
    @RequestMapping("/login")
    public ModelAndView login(@RequestParam("username") String username,
                              @RequestParam("password") String password,
                              HttpSession session,
                              HttpServletRequest request,
                              HttpServletResponse response) {
        ModelAndView mv = new ModelAndView();
        try {
            BookStoreServiceImpl service = new BookStoreServiceImpl(username, password);
            mv.setViewName("bookstore");

            //Session传值
            session.setAttribute(s: "bssi",service);
            return mv;
        } catch (Exception e) {
            mv.setViewName("index");
            mv.addObject(attributeName: "errorMessage",attributeValue: "登录错误! <br>"+ e.toString());
        }
        return mv;
    }
}
```

```java
    @RequestMapping("/buyBooks")
    public ModelAndView buyBooks(HttpSession session,
                                 @RequestAttribute("bookID") int bookID,
                                 @RequestAttribute("qty") int qty,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {
        ModelAndView mv = new ModelAndView();
        try {
            BookStoreServiceImpl bssi = (BookStoreServiceImpl) session.getAttribute(s: "bssi");
            bssi.buyBook(bookID, qty);
            mv.setViewName("bookstore");
            mv.addObject(attributeName: "message", attributeValue: "购买" + qty + "本书成功! ");
            session.setAttribute(s: "bssi", bssi);
            return mv;
        } catch (Exception e) {
            mv.setViewName("bookstore");
            mv.addObject(attributeName: "message", e.toString());
            return mv;
        }
    }
```

Session 传值

在用户登录时产生了构造服务类：BookStoreServiceImpl； 其中包含了可供用户操作的各种方法，所以在生成后要把这个类的实例传给下一个Controller，给用户提供持续的服务。这里使用了session传值



如果你不登录就访问打印书单的Controller 服务器将会返回NullPointerException 因为缺失了用户服务类的实例

02

使用JSP技术制作网页

2.1 页面跳转

2.2 前后台传值

2.3 页面表格生成

2.1 页面跳转 —— ModelAndView

```java
@RequestMapping("/login")
public ModelAndView login(@RequestParam("username") String username,
                          @RequestParam("password") String password,
                          HttpSession session,
                          HttpServletRequest request,
                          HttpServletResponse response) {
    ModelAndView mv = new ModelAndView();
    try {
        BookStoreServiceImpl service = new BookStoreServiceImpl(username, password);
        mv.setViewName("bookstore");

        //Session传值
        session.setAttribute( s: "bssi",service);
        return mv;
    } catch (Exception e) {
        mv.setViewName("index");
        mv.addObject( attributeName: "errorMessage", attributeValue: "登录错误！ <br>"+ e.toString());
    }
    return mv;
}
```

控制器根据JSP文件的存放位置进行页面跳转，由于之前在ServletConfig类中已经配置了JSP文件的前后缀，所以此处只用写文件名

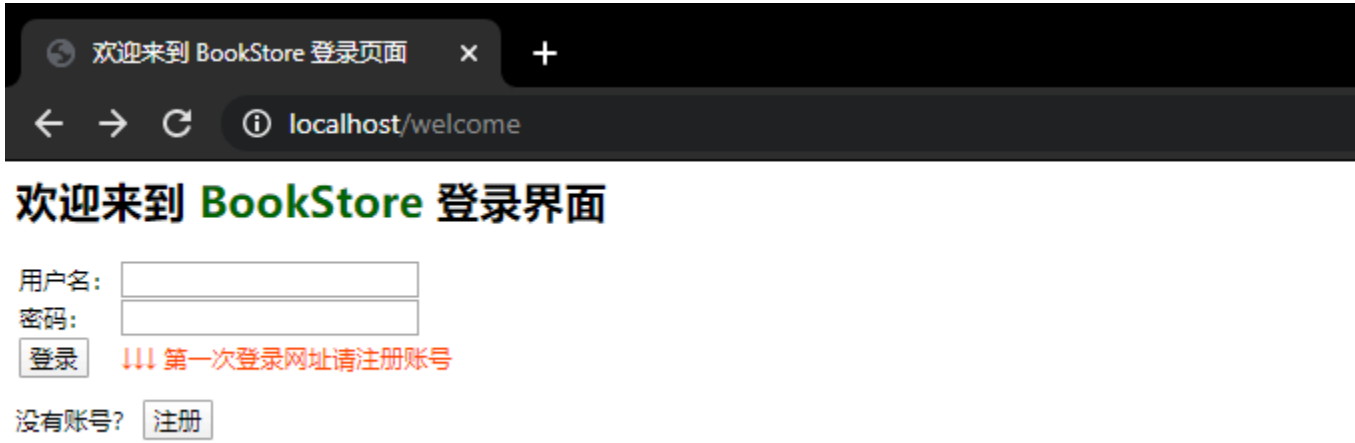
2.2 前后台传值 —— 前端给后端传值

```java
@RequestMapping("/doregs")
public ModelAndView doRegs(@RequestParam("username") String username,
                           @RequestParam("password") String password,
                           @RequestParam("sex") String Sex,
                           @RequestParam("birthday") String birthday,
                           HttpServletRequest request,
                           HttpServletResponse response) {
    ModelAndView mv = new ModelAndView();
    try {
        UserServiceImpl userService = new UserServiceImpl();
        userService.register(username, password, Sex, Date.valueOf(birthday), state: null);
        mv.setViewName("index");
        mv.addObject( attributeName: "errorMessage", attributeValue: "注册成功！现在可以登录啦\uD83D\uDE01");
        return mv;
    } catch (Exception e) {
        mv.setViewName("regs");
        mv.addObject( attributeName: "regsErrorMessage",("注册错误！<br>"+ e.toString()));
        return mv;
    }
}
```

http://localhost/regs/doregs?
username=username&
password=123&
sex=Male&
birthday=2019-12-18

前端传值给后端是在URL中进行传值，缺点是显示了明文，使用HTTP协议时很不安全

使用@RequestParam(“ElementID”)注解能够很方便地将指定的请求参数赋值给方法中的形参。



2.2 前后台传值 —— 后端前端给传值

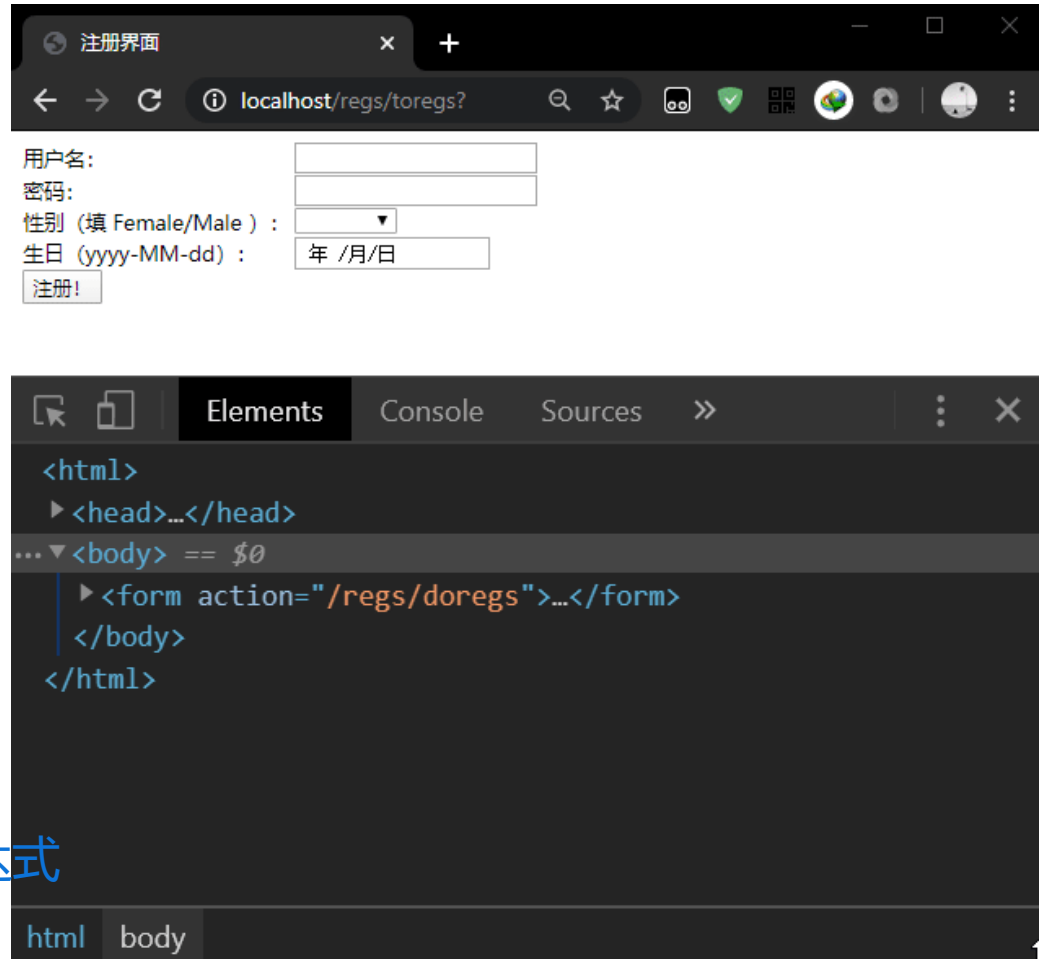
```java
@RequestMapping("/doregs")
public ModelAndView doRegs(@RequestParam("username") String username,
                           @RequestParam("password") String password,
                           @RequestParam("sex") String Sex,
                           @RequestParam("birthday") String birthday,
                           HttpServletRequest request,
                           HttpServletResponse response) {
    ModelAndView mv = new ModelAndView();
    try {
        UserServiceImpl userService = new UserServiceImpl();
        userService.register(username, password, Sex, Date.valueOf(birthday), state: null);
        mv.setViewName("index");
        mv.addObject( attributeName: "errorMessage", attributeValue: "注册成功！现在可以登录啦\uD83D\uDE01");
        return mv;
    } catch (Exception e) {
        mv.setViewName("regs");
        mv.addObject( attributeName: "regsErrorMessage",("注册错误！<br>"+ e.toString()));
        return mv;
    }
}
```

浏览器在request中获取文本

```jsp
<p style="color: orangered">
    <%=
        request.getAttribute("regsErrorMessage")
    %>
</p>
```

简化成EL表达式

```jsp
<td>
    <p style="color: orangered">${regsErrorMessage}</p>
</td>
```



通过EL表达式动态生成HTML

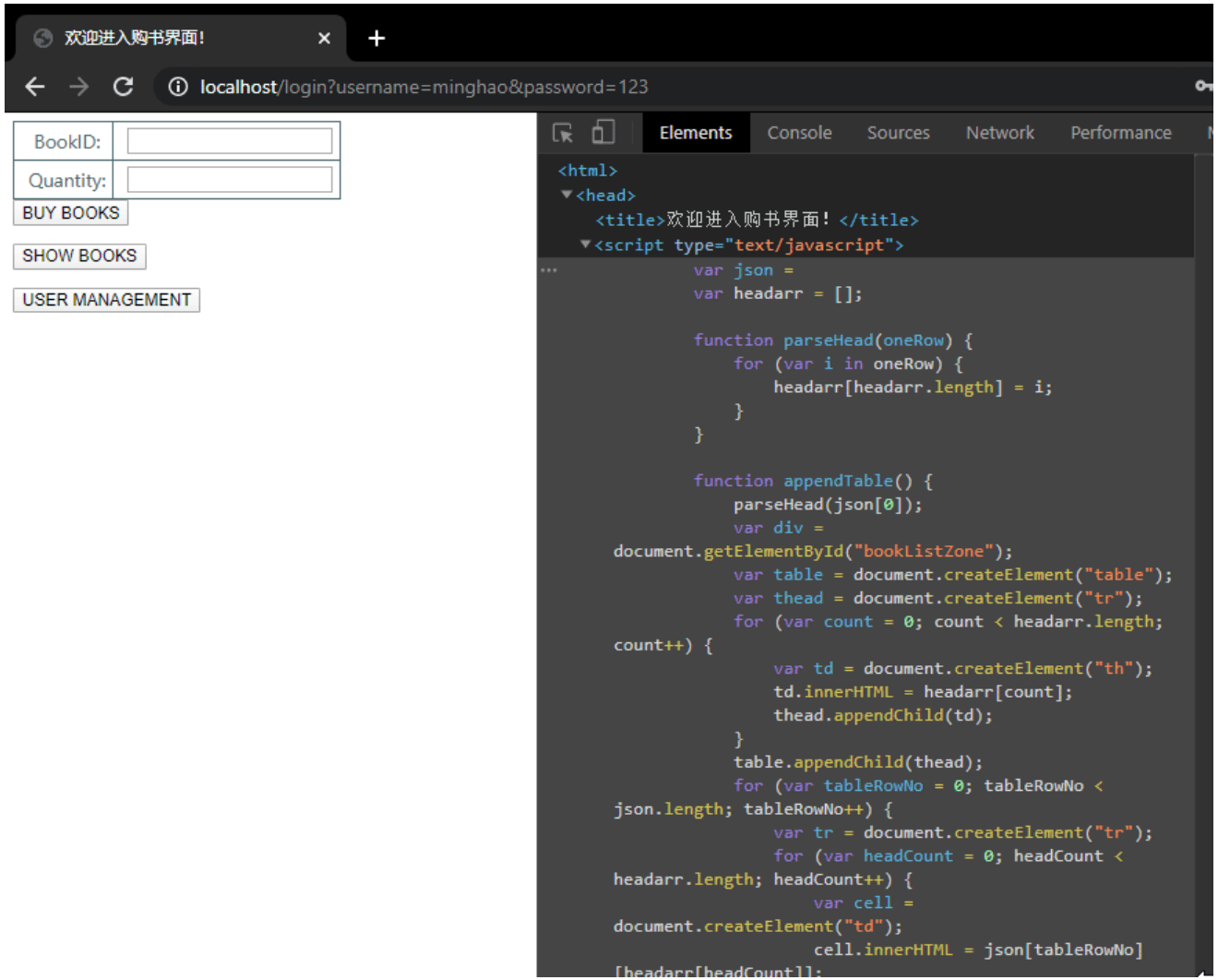
2.3 页面表格生成 —— 利用JSON

利用fastJSON把ObjectList序列化为适合HTTP传输的字符串

```java
@RequestMapping("/showBooks")
public ModelAndView showBooks(HttpSession session,
                              HttpServletRequest request,
                              HttpServletResponse response) {
    ModelAndView mv = new ModelAndView();
    try {
        BookStoreServiceImpl bssi = (BookStoreServiceImpl) session.getAttribute( s: "bssi");
        List<Book> bookList = bssi.getBookList();
        System.out.println(bookList);
        String jsonString = JSONObject.toJSONString(bookList);
        mv.setViewName("bookstore");
        mv.addObject( attributeName: "json", jsonString);
        session.setAttribute( s: "bssi", bssi);
        return mv;
    } catch (Exception e) {
        mv.setViewName("error");
        mv.addObject( attributeName: "message", e.toString());
        return mv;
    }
}
```

2.3 页面表格生成 —— 用于表格生成的JavaScript

```
<script type="text/javascript">
        var json =
        var headarr = [];

        function parseHead(oneRow) {
            for (var i in oneRow) {
                headarr[headarr.length] = i;
            }
        }

        function appendTable() {
            parseHead(json[0]);
            var div = document.getElementById("bookListZone");
            var table = document.createElement("table");
            var thead = document.createElement("tr");
            for (var count = 0; count < headarr.length; count++) {
                var td = document.createElement("th");
                td.innerHTML = headarr[count];
                thead.appendChild(td);
            }
            table.appendChild(thead);
            for (var tableRowNo = 0; tableRowNo < json.length; tableRowNo++) {
                var tr = document.createElement("tr");
                for (var headCount = 0; headCount < headarr.length;
headCount++) {
                    var cell = document.createElement("td");
                    cell.innerHTML = json[tableRowNo][headarr[headCount]];
                    tr.appendChild(cell);
                }
                table.appendChild(tr);
            }
            div.appendChild(table);

        }

</script>
```



实现传入JSON渲染表格的JavaScript

03
总结

3 总结

Spring MVC

1. 用MVC的模式编写代码

2. 体会AOP开发模式

3. Tomcat 服务器的配置

Hibernate

1. 利用ORM思想解决问题

2. 学会配置各种XML

3. 比较与普通JDBC的差别

用到的基本编程技术

Reflection	Annotation	Serialization	JDBC
反射	注解	序列化	数据库连接