第12章习题答案

一、

1.方法参数

2. HttpServletRequest对象、HttpServletResponse对象、HttpSession对象、Model/ModelMap对象

3. @RequestParam

4.八

5. HandlerMethodArgumentResolver

二、

1.C 2.A 3.B 4.A 5.D

三、

（1）答：前台页面通常会有数据提交给后台进行处理（如提交表单），参数绑定的作用就是将这些数据正确地交给Controller进行处理。

（2）答：在springMVC中提交请求的参数是通过方法的形参来接收的，从客户端请求的key/value数据，经过参数绑定，将key/value数据绑定到Controller的形参上，然后Controller就可以直接使用该形参了。

四、1.新建个person类：

1. package pojo;
2. public class Person {
3. private Integer id;
4. private String name;
5. private String tel;
6. private String address;
7. public Person() {
8. super();
9. }
10. public Person(Integer id, String name, String tel, String address) {
11. super();
12. this.id = id;
13. this.name = name;
14. this.tel = tel;
15. this.address = address;
16. }
17. public Integer getId() {
18. return id;
19. }
20. public void setId(Integer id) {
21. this.id = id;
22. }
23. public String getName() {
24. return name;
25. }
26. public void setName(String name) {
27. this.name = name;
28. }
29. public String getTel() {
30. return tel;
31. }
32. public void setTel(String tel) {
33. this.tel = tel;
34. }
35. public String getAddress() {
36. return address;
37. }
38. public void setAddress(String address) {
39. this.address = address;
40. }
41. @Override
42. public String toString() {
43. return "Person [id=" + id + ", name=" + name + ", tel=" + tel + ", address=" + address + "]";
44. }
45. }

2.编写page.jsp页面

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7. <title>page1</title>
8. </head>
9. <body>
10. <form action="person" method="get">
11. <label>学号：</label><input name="id"><br />
12. <label>姓名：</label><input name="name"><br />
13. <label>电话：</label><input name="tel"><br />
14. <label>地址：</label><input name="address"><br />
15. <input type="submit" value="确认">
16. </body>
17. </html>

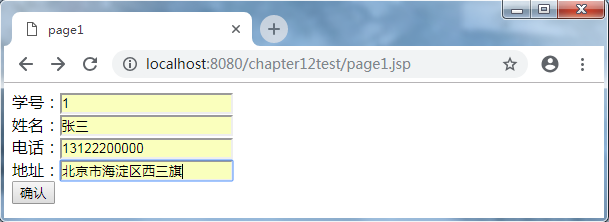
3.编写控制器类

1. package controller;
2. import java.io.IOException;
3. import javax.servlet.http.HttpServletResponse;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.web.bind.annotation.RequestMapping;
6. import org.springframework.web.bind.annotation.RequestMethod;
7. import pojo.Person;
8. @Controller
9. public class MyController {
10. @RequestMapping(value = "/person", method = RequestMethod.GET)
11. public String per(Person person,HttpServletResponse response) throws IOException {
12. System.out.println("POJO：" + person);
13. return "success";
14. }
15. }

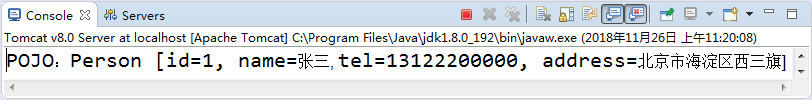
4.web.xml

1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" version="3.0">
5. <display-name>Archetype Created Web Application</display-name>
6. <!-- 编码过滤器 -->
7. <filter>
8. <filter-name>encodingFilter</filter-name>
9. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
10. <async-supported>true</async-supported>
11. <init-param>
12. <param-name>encoding</param-name>
13. <param-value>UTF-8</param-value>
14. </init-param>
15. </filter>
16. <filter-mapping>
17. <filter-name>encodingFilter</filter-name>
18. <url-pattern>/\*</url-pattern>
19. </filter-mapping>
20. <!-- Spring MVC servlet -->
21. <servlet>
22. <servlet-name>SpringMVC</servlet-name>
23. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
24. <init-param>
25. <param-name>contextConfigLocation</param-name>
26. <param-value>classpath:spring-mvc.xml</param-value>
27. </init-param>
28. <load-on-startup>1</load-on-startup>
29. <async-supported>true</async-supported>
30. </servlet>
31. <servlet-mapping>
32. <servlet-name>SpringMVC</servlet-name>
33. <!-- 此处可以可以配置成\*.do，对应struts的后缀习惯 -->
34. <url-pattern>/</url-pattern>
35. </servlet-mapping>
37. <welcome-file-list>
38. <welcome-file>/index.html</welcome-file>
39. <welcome-file>/index.jsp</welcome-file>
40. </welcome-file-list>
41. </web-app>

5．运行项目，浏览器中输入信息



6.控制台中打印输入信息



第13章习题答案

一、

1. resolveException

2. @ExceptionHandler

3. @controlleradvice @ ExceptionHandler

4. preHandle afterCompletion

5. 动态代理

二、

1.B 2.A 3.D 4.C 5.D

三、

（1）答：SpringMVC提供的异常处理有三种方式，一种是直接实现HandlerExceptionResolver接口，另一种是使用@ExceptionHandler注解实现一个专门用于处理异常的Controller，第三种是使用@ControllerAdvice注解，该注解使@ExceptionHandler可以监控到全局的异常。

（2）答：主要有两种方式： 第一种是实现HandlerInterceptor 接口，或者是继承实现了HandlerInterceptor 接口的类；第二种实现Spring的WebRequestInterceptor接口，或者是继承实现了WebRequestInterceptor的类。

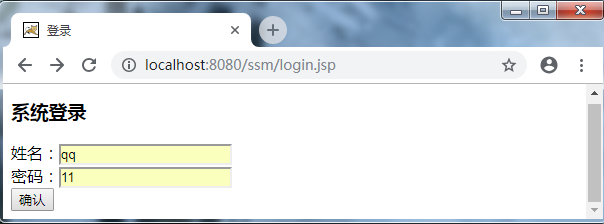
四、1.新建个拦截器类SpringMVCInterceptor，实现HandlerInterceptor

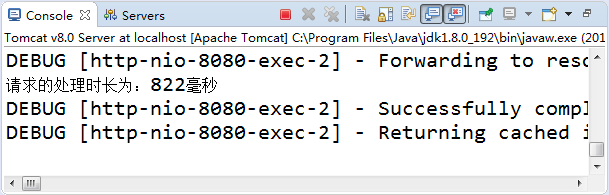
1. package com.qfedu.intercept;
2. import javax.servlet.http.HttpServletRequest;
3. import javax.servlet.http.HttpServletResponse;
4. import org.springframework.web.servlet.HandlerInterceptor;
5. import org.springframework.web.servlet.ModelAndView;
6. public class SpringMVCInterceptor implements HandlerInterceptor {
7. @Override
8. public boolean preHandle(HttpServletRequest request,
9. HttpServletResponse response, Object handler) throws Exception {
10. request.setAttribute("time", System.currentTimeMillis());
11. return true;
12. }
13. @Override
14. public void postHandle(HttpServletRequest request,
15. HttpServletResponse response, Object handler,
16. ModelAndView modelAndView) throws Exception {
17. }
18. @Override
19. public void afterCompletion(HttpServletRequest request,
20. HttpServletResponse response, Object handler, Exception ex)
21. throws Exception {
22. long time = (long)request.getAttribute("time");
23. System.out.println("请求的处理时长"+(System.currentTimeMillis() - time));
24. }
25. }

2.在Springmvc.xml配置文件中添加下列标签：

1. <!--配置拦截器 -->
2. <mvc:interceptors>
3. <mvc:interceptor>
4. <mvc:mapping path="/\*\*" />
5. <bean class="com.qfedu.intercept.SpringMVCInterceptor"/>
6. </mvc:interceptor>
7. </mvc:interceptors>

3.在浏览器中输入登录信息，点击确认，即可在控制台打印出请求时长。





第14章习题答案

一、

1. post

2. enctype

3. transferTo

4. @RequestBody @ResponseBody

5. 面向资源

二、

1.C 2. A 3.D 4.B 5.B

三、

（1）答; 静态资源在浏览器中无法加载，原因是web.xml文件中，配置了前端控制器使用的匹配规则为<url-pattern>/</url-pattern>，表示这个servlet 拦截了所有的请求，包括css,js等。这种情况下只要在SpringMVC的配置文件中添加<mvc:default-servlet-handler></mvc:default-servlet-handler>标签放行静态资源即可，还可以通过路径映射的方式，在SpringMVC的配置文件中设置静态资源映射，添加<mvc:resources>标签即可。

（2）答：通过Jackson框架就可以把Java里面的对象直接转化成js可以识别的Json对象 具体步骤如下 :

1.加入Jackson.jar

2.在配置文件中配置json的映射

3.在接受Ajax方法里面可以直接返回Object,List等,但方法前面要加上@ResponseBody注解

四、

文件上传控制器类：

1. @Controller
2. public class FileUpController {
3. @RequestMapping("fileup")
4. public void file(MultipartFile file, HttpServletRequest request, HttpServletResponse response)
5. throws IllegalStateException, IOException {
6. String fn = file.getOriginalFilename();
7. File desfile = new File(request.getSession().getServletContext().getRealPath("/"), fn);
8. file.transferTo(desfile);
9. response.getWriter().print(desfile.getAbsolutePath());
10. }
11. }

文件下载控制器类：

1. @Controller
2. public class FileDownController {
3. @RequestMapping("filedown")
4. public void down(HttpServletRequest request, HttpServletResponse response) throws IOException {
5. response.setContentType("text/html;charset=utf-8");
6. File file = new File(request.getSession().getServletContext().getRealPath("/"), "gs1.png");
7. System.out.println("文件" + file.getAbsolutePath());
8. byte[] data = FileUtils.readFileToByteArray(file);
9. response.setContentType("application/x-msdownload;");
10. response.setHeader("Content-disposition",
11. "attachment; filename=" + new String(file.getName().getBytes("utf-8"), "ISO8859-1"));
12. response.setHeader("Content-Length", String.valueOf(file.length()));
13. response.getOutputStream().write(data);
14. }
15. }

具体代码和配置详见本章案例14.1-14.5。

第15章习题答案

一、

1. Spring SpringMVC Mybatis

2. Mybatis

3. springmvc spring Mybatis

4. @RequestMapping

5. DispatcherServlet

二、

1.B 2.B 3.C 4.B 5.C

三、

（1）答：SpringMVC拥有控制器，作用跟Struts类似，接收外部请求，解析参数传给服务层；Spring容器属于协调上下文，管理对象间的依赖，提供事务机制；Mybatis属于orm持久层框架，将业务实体与数据表联合起来。

（2）答：1.方便解耦，简化开发，Spring就像是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理。 2.AOP编程的支持，Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能。3.声明式事务的支持，只需要通过配置就可以完成对事务的管理，而无需手动编程。 4.方便程序的测试，Spring对Junit4支持，可以通过注解方便的测试Spring程序。5.降低JavaEE API的使用难度，Spring 对JavaEE开发中非常难用的一些API（JDBC、JavaMail、远程调用等），都提供了封装，使这些API应用难度大大降低。

四、

1．新建数据库ssm、students数据表：

1. DROP DATABASE IF EXISTS ssm;
2. CREATE DATABASE ssm;
3. use ssm;
4. CREATE TABLE `students` (
5. `id` int(20) NOT NULL ,
6. `name` varchar(20) NOT NULL ,
7. `password` varchar(20) NOT NULL ,
8. `sex` varchar(8) NOT NULL ,
9. `age` int(8) NOT NULL ,
10. `clazz` varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL
11. )

2.新建student类：

1. package com.qfedu.pojo;
2. public class Student {
3. private int id;
4. private String name;
5. private String password;
6. private String sex;
7. private int age;
8. private String clazz;
9. public Student() {
10. super();
11. // TODO Auto-generated constructor stub
12. }
13. public Student(int id, String name, String password, String sex, int age, String clazz) {
14. super();
15. this.id = id;
16. this.name = name;
17. this.password = password;
18. this.sex = sex;
19. this.age = age;
20. this.clazz = clazz;
21. }
22. public int getId() {
23. return id;
24. }
25. public void setId(int id) {
26. this.id = id;
27. }
28. public String getName() {
29. return name;
30. }
31. public void setName(String name) {
32. this.name = name;
33. }
34. public String getPassword() {
35. return password;
36. }
37. public void setPassword(String password) {
38. this.password = password;
39. }
40. public String getSex() {
41. return sex;
42. }
43. public void setSex(String sex) {
44. this.sex = sex;
45. }
46. public int getAge() {
47. return age;
48. }
49. public void setAge(int age) {
50. this.age = age;
51. }
52. public String getClazz() {
53. return clazz;
54. }
55. public void setClazz(String clazz) {
56. this.clazz = clazz;
57. }
58. @Override
59. public String toString() {
60. return "Student [id=" + id + ", name=" + name + ", password=" + password + ", sex=" + sex + ", age=" + age
61. + ", clazz=" + clazz + "]";
62. }
63. }

3．编写dao：

1. package com.qfedu.dao;
2. import com.qfedu.pojo.Student;
3. public interface StudentDao {
4. //添加学生信息
5. int insertStu(Student student);
6. //根据姓名登录
7. Student selectByName(String name);
8. }

4.编写mapper文件：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3. <!--命名空间应该是对应接口的包名+接口名 -->
4. <mapper namespace="com.qfedu.dao.StudentDao">
5. <!--添加学生信息 -->
6. <insert id="insertStu" >
7. insert into students(id,name,password,sex,age,clazz)
8. values(#{id},#{name},#{password},#{sex},#{age},#{clazz})
9. </insert>
10. <!--根据姓名查询学生信息 -->
11. <select id="selectByName" resultType="com.qfedu.pojo.Student">
12. select id,name,password,sex,age,clazz
13. from students where name=#{name}
14. </select>
15. </mapper>

5.编写service

1. package com.qfedu.service;
2. import com.qfedu.pojo.Student;
3. public interface StudentService {
4. //学生登录功能
5. Student loginStu(String name,String password);
6. //学生注册功能
7. int addStu(Student student);
8. }

6.新建小配置文件db.properties:

1. mysql.driver=com.mysql.jdbc.Driver
2. mysql.url=jdbc:mysql://localhost:3306/ssm
3. mysql.uid=root
4. mysql.password=root
5. mysql.acquireIncrement=5
6. mysql.initialPoolSize=10
7. mysql.minPoolSize=5
8. mysql.maxPoolSize=20

7.spring.xml配置文件信息：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:aop="http://www.springframework.org/schema/aop"
5. xmlns:context="http://www.springframework.org/schema/context"
6. xmlns:tx="http://www.springframework.org/schema/tx"
7. xsi:schemaLocation="http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
9. http://www.springframework.org/schema/context
10. http://www.springframework.org/schema/context/spring-context-4.3.xsd
11. http://www.springframework.org/schema/aop
12. http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
13. http://www.springframework.org/schema/tx
14. http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">
15. <context:component-scan base-package="com.qfedu.\*" />
16. <!--1 引入属性文件，在配置中占位使用 -->
17. <context:property-placeholder location="classpath:db.properties" />
18. <!--2 配置C3P0数据源 -->
19. <bean id="datasource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
20. destroy-method="close">
21. <!--驱动类名 -->
22. <property name="driverClass" value="${mysql.driver}" />
23. <!-- url -->
24. <property name="jdbcUrl" value="${mysql.url}" />
25. <!-- 用户名 -->
26. <property name="user" value="${mysql.uid}" />
27. <!-- 密码 -->
28. <property name="password" value="${mysql.password}" />
29. <!-- 当连接池中的连接耗尽的时候c3p0一次同时获取的连接数 -->
30. <property name="acquireIncrement" value="${mysql.acquireIncrement}"></property>
31. <!-- 初始连接池大小 -->
32. <property name="initialPoolSize" value="${mysql.initialPoolSize}"></property>
33. <!-- 连接池中连接最小个数 -->
34. <property name="minPoolSize" value="${mysql.minPoolSize}"></property>
35. <!-- 连接池中连接最大个数 -->
36. <property name="maxPoolSize" value="${mysql.maxPoolSize}"></property>
37. </bean>
38. <!--3 会话工厂bean sqlSessionFactoryBean -->
39. <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
40. <!-- 数据源 -->
41. <property name="dataSource" ref="datasource"></property>
42. <property name="configLocation" value="classpath:mybatis-config.xml"></property>
43. <!-- sql映射文件路径 -->
44. <property name="mapperLocations" value="classpath\*:com/qfedu/dao/impl/\*Mapper.xml"></property>
45. </bean>
46. <!-- 自动扫描对象关系映射 -->
47. <!--创建dao-->
48. <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
49. <!--指定会话工厂，如果当前上下文中只定义了一个则该属性可省去 -->
50. <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"></property>
51. <!-- 指定要自动扫描接口的基础包，实现接口 -->
52. <property name="basePackage" value="com.qfedu.dao"></property>
53. </bean>
54. </beans>

8.编写serviceImpl：

1. package com.qfedu.service.impl;
2. import javax.annotation.Resource;
3. import org.springframework.stereotype.Service;
4. import com.qfedu.dao.StudentDao;
5. import com.qfedu.pojo.Student;
6. import com.qfedu.service.StudentService;
7. @Service
8. public class StudentServiceImpl implements StudentService{
9. @Resource
10. private StudentDao studentDao;
11. @Override
12. public Student loginStu(String name, String password) {
13. Student stu = studentDao.selectByName(name);
14. if(stu!=null&&stu.getPassword().equals(password)) {
15. return stu;
16. }else {
17. return null;
18. }
19. }
20. @Override
21. public int addStu(Student student) {
22. return studentDao.insertStu(student);
23. }
24. }

9.配置springmvc.xml配置文件：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:mvc="http://www.springframework.org/schema/mvc"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans.xsd
8. http://www.springframework.org/schema/context
9. http://www.springframework.org/schema/context/spring-context.xsd
10. http://www.springframework.org/schema/mvc
11. http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12. <!-- 启动自动扫描 -->
13. <context:component-scan
14. base-package="com.qfedu.controller" />
15. <!-- 注册MVC注解驱动 -->
16. <mvc:annotation-driven />
17. <!-- 静态资源可访问的设置方式 -->
18. <mvc:default-servlet-handler />
19. <!-- 配置视图解析器，可以显式设置，也可以不设置，不设置会依据SpringMVC的默认设置 -->
20. <bean id="viewResolver"
21. class="org.springframework.web.servlet.view.InternalResourceViewResolver">
22. <property name="prefix" value="/" />
23. <property name="suffix" value=".jsp" />
24. </bean>
25. </beans>

10.编写登录和注册控制器类：

1. @Controller
2. public class LoginController {
3. @Resource
4. private StudentService StudentService;
5. @RequestMapping(value="/login")
6. public String loginStu(String name, String password,HttpServletResponse response) {
7. Student stu = StudentService.loginStu(name, password);
8. if(stu!=null) {
9. return "success";
10. }else {
11. return "error";
12. }
13. }
14. }
15. @Controller
16. public class AddController {
17. @Resource
18. private StudentService StudentService;
19. @RequestMapping(value="/add")
20. public String addStu(Student stu, HttpServletResponse response) {
21. int n = StudentService.addStu(stu);
22. if(n>=0) {
23. return "success";
24. }else {
25. return "error";
26. }
27. }
28. }

11.配置web.xml：

1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4. xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5. http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">
6. <display-name>Archetype Created Web Application</display-name>
7. <!--编码格式过滤器-->
8. <filter>
9. <filter-name>encoding</filter-name>
10. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
11. <init-param>
12. <param-name>encoding</param-name>
13. <param-value>UTF-8</param-value>
14. </init-param>
15. </filter>
16. <filter-mapping>
17. <filter-name>encoding</filter-name>
18. <url-pattern>/\*</url-pattern>
19. </filter-mapping>
20. <!--Spring的配置文件-->
21. <context-param>
22. <param-name>contextConfigLocation</param-name>
23. <param-value>classpath:spring.xml</param-value>
24. </context-param>
25. <!--Spring的监听器-->
26. <listener>
27. <listener-class>
28. org.springframework.web.context.ContextLoaderListener
29. </listener-class>
30. </listener>
31. <!--SpringMVC的前端控制器-->
32. <servlet>
33. <servlet-name>springMVC</servlet-name>
34. <servlet-class>
35. org.springframework.web.servlet.DispatcherServlet
36. </servlet-class>
37. <init-param>
38. <param-name>contextConfigLocation</param-name>
39. <param-value>classpath:spring-mvc.xml</param-value>
40. </init-param>
41. <load-on-startup>1</load-on-startup>
42. </servlet>
43. <!--访问DispatcherServlet对应的路径-->
44. <servlet-mapping>
45. <servlet-name>springMVC</servlet-name>
46. <url-pattern>/</url-pattern>
47. </servlet-mapping>
48. </web-app>

12.编写登录注册以及操作成功或失败的跳转页面：

注册页面，add.jsp:

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7. <title>注册</title>
8. </head>
9. <body>
10. <h3>注册学生信息</h3>
11. <form action="add">
12. <label>学号：</label><input name="id"><br />
13. <label>姓名：</label><input name="name"><br />
14. <label>密码：</label><input name="password"><br />
15. <label>性别：</label><input name="sex"><br />
16. <label>年龄：</label><input name="age"><br />
17. <label>爱好课程：</label><input name="clazz"><br />
18. <input type="submit" value="确认">
19. </form>
20. </body>
21. </html>

登录页面，login.jsp:

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7. <title>登录</title>
8. </head>
9. <body>
10. <h3>系统登录</h3>
11. <form action="login">
12. <label>姓名：</label><input name="name"><br />
13. <label>密码：</label><input name="password"><br />
14. <input type="submit" value="确认">
15. </form>
16. </body>
17. </html>

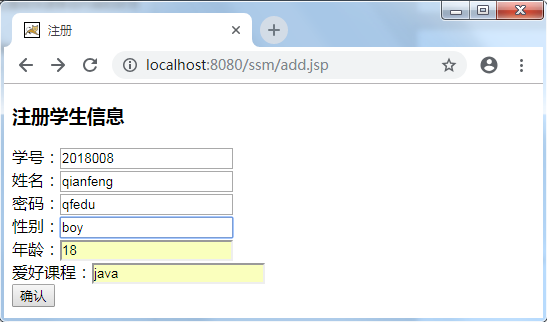
错误页面，error.jsp:

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7. <title>服务器暂时无法访问</title>
8. </head>
9. <body>
10. <h2>error</h2>
11. <h4 style="color: red">本次操作失败！</h4>
12. </body>
13. </html>

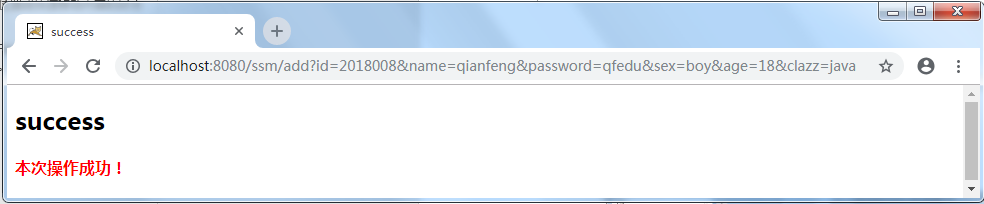
操作成功页面，success.jsp：

1. <%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7. <title>服务器暂时无法访问</title>
8. </head>
9. <body>
10. <h2>error</h2>
11. <h4 style="color: red">本次操作失败！</h4>
12. </body>
13. </html>

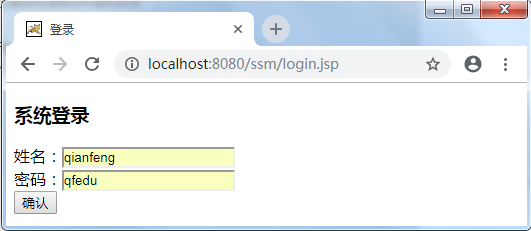
完成代码的编写后，在浏览器访问，并注册学生信息：



操作成功后会跳转至success.jsp页面，插入的信息可以从地址栏中看到：



在浏览器输入登录页面地址，完成注册信息的登录：



如果密码输入正确即可跳转至成功页面：

![C:\Users\Administrator\AppData\Roaming\Tencent\Users\1045893715\QQ\WinTemp\RichOle\O7AY@](8I`{_`~]79$KY5J6.png](data:image/png;base64,)