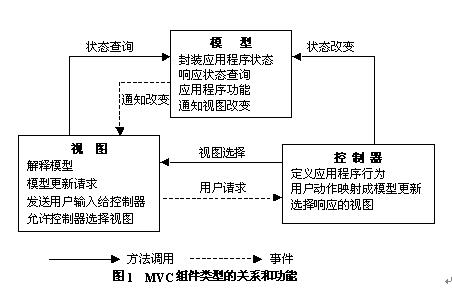
Spring MVC

1. MVC模式介绍

模型-视图-控制器(Model-View-Controller)



模型：实体Bean、Entity、VO、FormBean

控制器：Servlet、Struts、Spring framework

视图：JSP、HTML、(控制器可以直接返回XML,json数据给客服端)

把一个应用的输入、处理、输出流程按照Model、View、Controller的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

MVC模式是解决了展示以及业务逻辑实现的分离。让开发者

可以专心地解决不同层的编码。也方便代码重用和测试和维护。

1. MVC设计思想

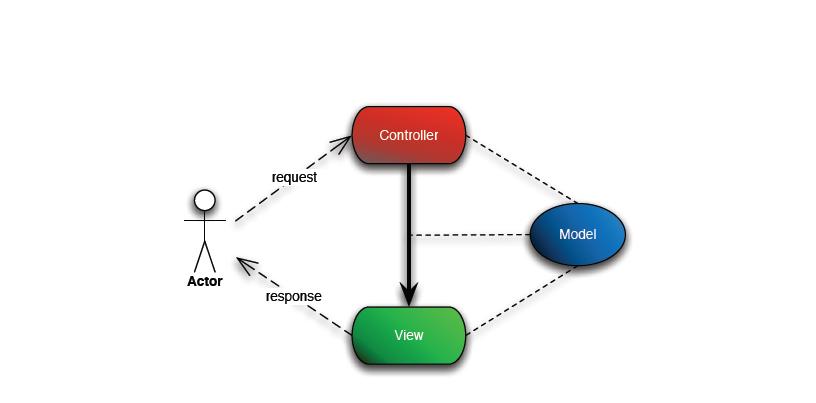
MVC是一种复合模式，结合了观察者模式、策略模式、组合模式、适配器模式。

模型使用观察者模式，以便观察者更新，同时保持两者之间的解耦。控制器是视图的策略，视图使用组合模式实现用户界面。适配器模式用来将模型适配成符合现有视图和控制器的需要的模型。

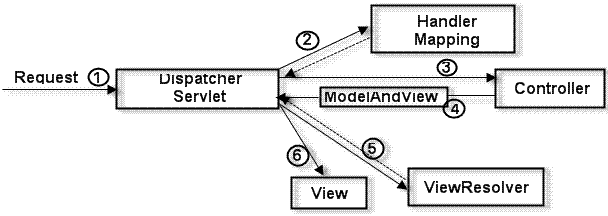
这些模式合作把MVC模式的三层解耦。

1. Spring MVC架构

Spring MVC是结构最清晰的MVC Model 2实现。它的Action也不叫Action，而是称做Controller；Controller接收request, response参数，然后返回ModelAndView（其中的Model不是Object类型，而是Map类型）。但在其它的Web Framework中，Action返回值一般都只是一个View Name；Model则需要通过其它的途径（如request.attribute，Context参数，或Action本身的属性数据）传递上去。



1. Spring MVC重要组件



1、DispatcherServlet：前端控制器，用于请求到达前端控制器，由它调用其他组件处理用户的请求，相当于转发器。DispatcherServlet是Spring mvc的中央调度器，DispatcherServlet创建时会默认从DispatcherServlet.properties文件加载Springmvc所用的各种组件。

2、HandlerMapping：处理器映射器，负责根据用户请求找到Controller(处理器)，springmvc提供了不同的映射器实现方式。处理器映射器负责根据request请求找到对应的Controller处理器以及Interceptor拦截器，将它们封装在HandlerExecutionChain对象中返回给前端控制器。

3、Controller：处理器，对具体的用户请求进行处理。

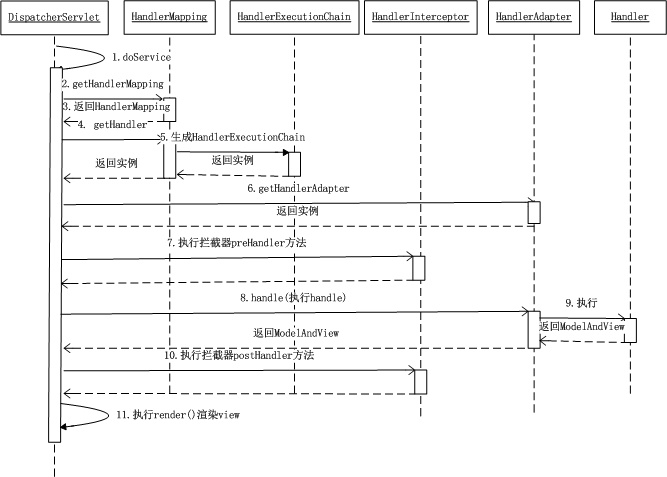
4、View Resolver：视图解析器，负责将处理结果生成view视图。View Resolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。

5、View:View是一个接口，实现类支持不同的View类型，springmvc框架提供了很多的View视图类型，包括：jstlView、freemarkerView、pdfView等。

1. Spring MVC重要组件分析[含代码和实现界面]
2. 代码说明

SpringMVC的学习例子，使用IntelliJ IDEA进行的开发的，Spring Framework，hibernate，mysql, JPA 的web项目。本项目是一个完全阐述Spring MVC工作流程和原理的例子，包含了一个只有两张表的数据库；一和controller，实现增删改查(CRUD)

2、Spring MVC的流程图：



3、Spring MVC的主要工作流程描述：

（1）用户向服务器(Tomcat,Apache,etc...)发送HTTP请求，请求被Spring MVC 前段控制器DispatcherServlet捕获。然后通过web.xml里面的url映射关系,把需要处理的请求交给后台Controller处理,此案例中把所有访问/目录下的请求都交给Controlloer处理,

DispatcherServlet在web.xml中的配置：

<servlet>

<servlet-name>mvc-dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>mvc-dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<load-on-startup>1</load-on-startup>：表示启动Web服务器时优先初始化该Servlet；

url-pattern：映射需要被控制器处理的请求url,“/” 表示该主机下所有资源都被DispatcherServlet所拦截。也可以如“\*.html”表示拦截所有以html为扩展名的请求。

启动web服务器,部署项目到Tomcat服务器的webapps目录下，输入“localhost:8080/”回车即向服务器发送了一个请求。

（2）DispatcherServlet对请求URL进行解析，得到请求资源标识符（URI）。然后根据该URI，调用HandlerMapping获得该Handdler配置的所有相关的对象（包括Handdler对象以及Handdler对象对应的拦截器），最后以HandlerExecutionChain对象的形式返回；

示例WEB-INF\ mvc-dispatcher-servlet.xml：

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:jpa="http://www.springframework.org/schema/data/jpa"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">

<!--指明 controller 所在包，并扫描其中的注解-->

<context:component-scan base-package="com.cenyol.example.controller"/>

（3）DispatcherServlet将请求提交到目标Controller。

（4）Controller进行业务逻辑处理后，向DispatcherServlet返回一个ModelAndView。

<!--ViewResolver 视图解析器-->

<!--用于支持Servlet、JSP视图解析-->

<bean id="jspViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>

<property name="prefix" value="/html/"/>

<property name="suffix" value=".jsp"/>

</bean>

对于每个请求@RequestMapping都返回的仅是一个没有后缀名的文件，因为视图解析器会自动添加前缀路径和后缀.jsp,服务器容器自动将.jsp文件翻译成.html文件返回给客户端。以下Controller代码定义了各种CRUD请求，

MainController.java:

package com.cenyol.example.controller;

import com.cenyol.example.model.UserEntity;

import com.cenyol.example.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

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

import org.springframework.web.bind.annotation.RequestMethod;

import java.lang.reflect.Method;

import java.util.List;

@Controller

public class MainController {

// 自动装配

@Autowired

private UserRepository userRepository;

// 首页

@RequestMapping(value = "/", method = RequestMethod.GET)

public String index() {

return "index";

}

// 用户管理

@RequestMapping(value = "/users", method = RequestMethod.GET)

public String users(ModelMap modelMap){

// 找到user表里面的所有记录

List<UserEntity> userEntityList = userRepository.findAll();

// 将所有的记录传递给返回的jsp页面

modelMap.addAttribute("userList", userEntityList);

// 返回pages目录下的userManage.jsp

return "userManage";

}

// 添加用户表单页面

@RequestMapping(value = "/addUser", method = RequestMethod.GET)

public String addUser(){

return "addUser";

}

// 添加用户处理

@RequestMapping(value = "/addUserPost", method = RequestMethod.POST)

public String addUserPost(@ModelAttribute("user") UserEntity userEntity){

// 向数据库添加一个用户

//userRepository.save(userEntity);

// 向数据库添加一个用户，并将内存中缓存区的数据刷新，立即写入数据库，之后才可以进行访问读取

userRepository.saveAndFlush(userEntity);

// 返回重定向页面

return "redirect:/users";

}

// 查看用户详细信息

// @PathVariable可以收集url中的变量，需匹配的变量用{}括起来

// 例如：访问 localhost:8080/showUser/1 ，将匹配 userId = 1

@RequestMapping(value = "/showUser/{userId}", method = RequestMethod.GET)

public String showUser( @PathVariable("userId") Integer userId, ModelMap modelMap ){

UserEntity userEntity = userRepository.findOne(userId);

modelMap.addAttribute("user", userEntity);

return "userDetail";

}

// 更新用户信息页面

@RequestMapping(value = "/updateUser/{userId}", method = RequestMethod.GET)

public String updateUser(@PathVariable("userId") Integer userId, ModelMap modelMap){

UserEntity userEntity = userRepository.findOne(userId);

modelMap.addAttribute("user", userEntity);

return "updateUser";

}

// 处理用户修改请求

@RequestMapping(value = "/updateUserPost", method = RequestMethod.POST)

public String updateUserPost(@ModelAttribute("user") UserEntity userEntity){

userRepository.updateUser(

userEntity.getFirstName(),

userEntity.getLastName(),

userEntity.getPassword(),

userEntity.getId()

);

return "redirect:/users";

}

// 删除用户

@RequestMapping(value = "/deleteUser/{userId}", method = RequestMethod.GET)

public String deleteUser(@PathVariable("userId") Integer userId){

// 删除id为userId的用户

userRepository.delete(userId);

// 立即刷新数据库

userRepository.flush();

return "redirect:/users";

}

}

（5）Dispathcher查询一个或多个ViewResolver视图解析器进行视图解析,找到ModelAndView对象指定的视图对象。

WEB-INF\ mvc-dispatcher-servlet.xml中ViewResolver 视图解析器：

<!--ViewResolver 视图解析器-->

<!--用于支持Servlet、JSP视图解析-->

<bean id="jspViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>

<property name="prefix" value="/html/"/>

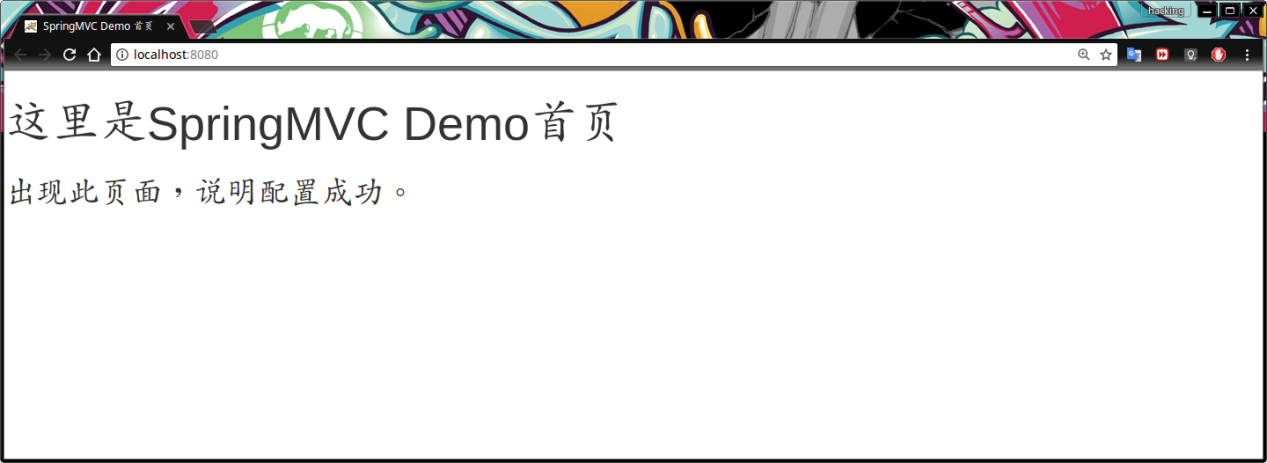
<property name="suffix" value=".jsp"/>

</bean>

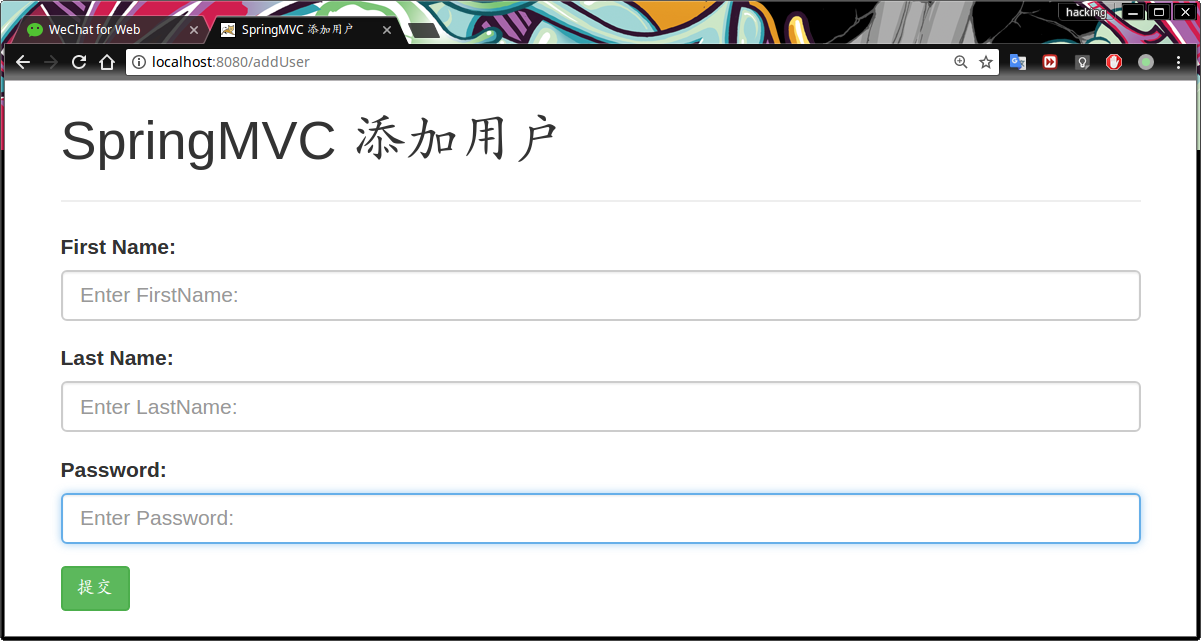
（6）视图对象负责渲染返回给客户端。

4、运行结果：

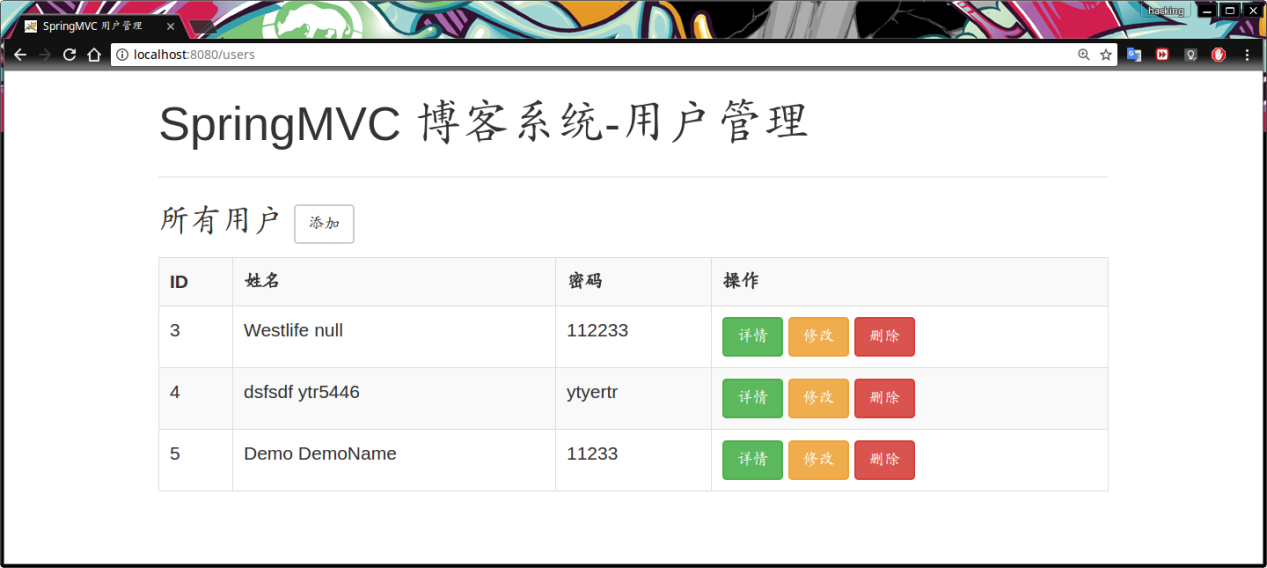
（1）请求成功后



(2)用户信息添加



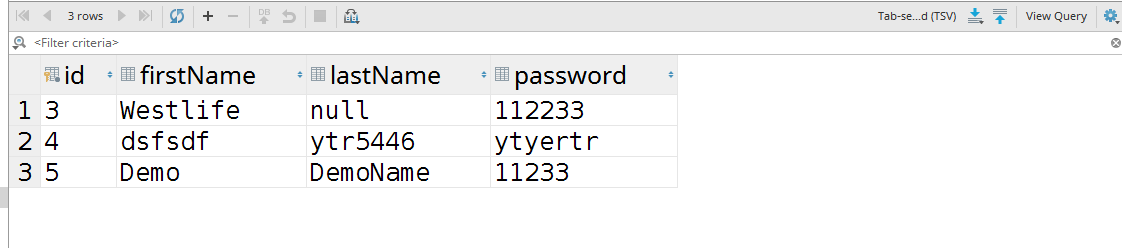
（3）用户管理界面



（4）用户详情



（5）数据库详情



六、总结

优点：

1.视图控制模型分离， 提高代码重用性

2.提高开发效率

3.便于后期维护， 降低维护成本

4.方便多开发人员间的分工

缺点:

1.清晰的构架以代码的复杂性为代价， 对小项目优可能反而降低开发效率

2.运行效率相对较低

3.目前没有比较好的rich 客户端的解决方案

4.控制层和表现层有时会过于紧密，导致没有真正分离和重用