北风教育培训中心指定教材

BF-TECH 5.0 软件工程师培训指定用书

SpringMVC+MyBatis 高级框架技术

上海育创网络科技有限公司　编著

Shanghai YuChuang Infomation Technology Ltd.

书　　名：SpringMVC+MyBatis 高级框架技术

版 本 号：BF-TECH 5.0

总 策 划：童金浩

责任编辑：娄景亮

研发成员：陈凯（组长）卿小春 霍谅 史良 戴彬 唐徽

责任校对：唐徽

编辑单位：上海育创网络科技有限公司

邮　　编：200122

网　　站：www.ibeifeng.com

印　　数：1000 册

版　　次：2016 年 3 月第 1 版

印　　次：2016 年 3 月第 1 次印刷

规　　格：A4（21cm*29.7cm）

前 言

北风网创办于 2008 年，是一家 IT 在线教育培训互联网公司。公司以就业、跳槽为导向，以互联网为平台；以“打造中国在线职业教育领导品牌、构建互联网职业教育绿色生态圈”为公司的愿景；以“教育创造美好未来：为客户提供持续就业的竞争能力，为员工提供快乐、成长的内部创业平台，为企业输送高品质人才，提升企业核心竞争力”为公司使命，为在校大学生、IT 从业者提供高性价比的 IT 实战培训，持续提升学员的职场竞争能力，实现学员整个职业生涯“从菜鸟到事业合伙人”的全面提升解决方案。公司至今已经获得两轮融资，在同行业中处于领先地位。

BF-TECH 是北风网自 2012 年以来精心打造的一系列品牌课程。本系列课程推出后，以课程技术的前沿性，项目实战贴近企业需求等鲜明特色，很快引发了一场 IT 在线教育的风暴，极大的冲击了市场上在线教育和实体教育的同类产品，并获得了一大批理解、认同、追随 BF-TECH 系列品牌的忠实客户群体。在追求品质的道路上，北风产品人从未停止过前进的脚步，我们不满足于现有成果的取得，在原有的 BF-TECH 4.0 课程的基础之上，北风产品研发团队，在经过与多位软件领域专家和资深软件工程师的深入探讨，并结合大数据技术，对主流的企业招聘网站近万份招聘数据的分析，历经近 1 年左右时间的产品研发，BF-TECH 5.0 系列品牌就业课程终于成功面世。本系列课程关注 IT 行业技术发展驱势，聚焦企业正在使用和未来将要流行的热门技术，为有志于从事 IT 软件开发工程师工作岗位的人员提供了一座桥梁。

为了开发这套课程，北风的教研团队，技术团队，市场团队，就业团队 均加入了我们课程研发队伍，北风的合作企业与签约讲师也为我们提供了大量的企业项目实训素材，我们秉承着“做有情怀的公司，做有灵魂的产品”的产品理念，对我们教学产品精益求精，我们团队每个人付出不亚于任何人的努力，我们相信，产品助力 – 终将会让教学回归本质，我们坚信，教育一定能创建美好未来！

童金浩

内容简介

本套课程由北风大数据讲师团队和 Java 讲师团队，反复的市场调研与论证，并由众多一线兼职大数据讲师与 Java 讲师参与强强联手合作，打造的 “国内顶级零基础大数据就业课程“，4 大阶段，13 门课程，3+ 大数据项目实战。内容涵盖了 J2EE、HTML5、SpringMVC、Mybatis、Hadoop 2.x、Spark 和 Storm 等多种技术，旨在培训真正的 JAVA 开发和大数据应用高素质复合型人才。

Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制；MyBatis 是一个基于 Java 的持久层框架。

本书内容以 SpringMVC 和 MyBatis 高级框架技术两个大部分为主，其下包含了 springmvc 整体架构详解以及处理请求的流程；springmvc 映射器、师徒解析器详解；springmvc 传递参数、访问静态资源；springmvc 处理 json 数据；springmvc 实现文件上传；mybatis 环境搭建与基础知识点详解；mybatis 实现的增删改查操作；mybatis 与 springmvc 集成环境等。内容丰富，并附以案例及作业，使学生能更好的掌握知识。

通过本课程的学习，学员能够完成以下各个目标：

- ✧ 搭建 Spring MVC 环境
- ✧ 完成 Controller 和 Viewer 的映射
- ✧ 掌握异常处理
- ✧ 使用 Servlet API 对象作为入参
- ✧ Spring MVC 访问静态文件
- ✧ 掌握文件上传处理
- ✧ 掌握 Spring MVC 返回 JSON 数据
- ✧ 掌握 MyBatis 核心类的作用域以及生命周期
- ✧ 熟练运用 MyBatis 框架进行开发
- ✧ 掌握与 Spring 框架集成
- ✧ 使用 SSM 或 SpringMVC+MyBatis 框架开发项目

零基础大数据就业课程

SpringMVC+MyBatis 高级框架技术

做有情怀的公司，做有灵魂的产品

市场导向，质量为本，创新为魂，产品为根，服务为王，精益求精

目录

目录 7
前言（课程导学） 12
一、课程安排 12
二、课程学习目标 12
三、前导课程 12
四、后继课程 13
第一章 SpringMVC 框架 14
学习任务 1：Spring MVC 框架-1 14
回顾与作业点评 14
本章任务 14
本章目标 14
本章内容（学习活动） 14
1.1 Spring MVC 架构 14
1.2 Spring MVC 处理请求流程 15
1.3 Spring MVC 框架模型 15
1.4 Spring MVC 特点 15
1.5 Hello,SpringMVC-1 16
1.6 Hello,SpringMVC-2 16
1.7 HandlerMapping 17
1.8 视图解析器 18
1.9 参数传递-1 18
1.10 参数传递-2 19
1.11 使用 JSTL 实现页面输出 20
1.12 REST 风格 21

1.13 Spring　MVC 实现用户管理-1……21
1.14 Spring　MVC 实现用户管理-2……22
1.15 Spring　MVC 实现用户管理-3……22
1.16 Spring　MVC 实现用户管理-4……23
本章总结……24
本章作业……24
作业 1、Hello,SpringMVC-1……24
作业 2、Hello,SpringMVC-2……24
作业 3、参数传递-把值传给 Controller……25
作业 4、参数传递- Controller 把值传给 View……25
作业 5、使用 JSTL 标签实现页面输出……25
作业 6、SpringMVC 实现用户管理之用户列表……25
作业 7、SpringMVC 实现用户管理之增加用户……26
作业 8、SpringMVC 实现用户管理之修改用户……26
作业 9、SpringMVC 实现用户管理之删除用户……26
学习任务 2：Spring MVC 框架-2……27
回顾与作业点评……27
本章任务……27
本章目标……27
本章内容（学习活动）……27
1.1 Spring　MVC 实现用户登录……27
1.2 静态资源文件处理……28
1.3 处理文件上传-单文件上传……28
1.4 处理文件上传-多文件上传……29
1.5 Spring　MVC 返回 JSON 数据……30
本章总结……31
本章作业……32

作业 1、SpringMVC 实现用户管理之用户登录……32
作业 2、静态资源文件的引用……32
作业 3、处理文件上传-单文件上传……32
作业 4、处理文件上传-多文件上传……33
作业 5、Spring MVC 返回 JSON 数据……33
第二章 MyBatis 框架……34
学习任务 1：初识 MyBatis……34
本章目标……34
本章内容（学习活动）……34
1.1 MyBatis 简介……34
1.2 搭建 MyBatis 开发环境……34
1.3 查询用户表记录数……35
1.4 与 JDBC 直观对比……35
1.5 与传统 JDBC 的比较……36
1.6 与 Hibernate 的比较……36
1.7 MyBatis 基本要素……36
1.8 核心接口和类的结构……36
1.9 核心对象……37
1.10 SqlSessionFactoryBuilder-1……37
1.11 SqlSessionFactoryBuilder-2……37
1.12 SqlSessionFactoryBuilder-2……38
1.13 SqlSessionFactory……38
1.14 SqlSession-1……38
1.15 SqlSession-2……38
1.16 SqlSession-3……39
1.17 configuration.xml……40
1.18 properties……40

1.19 Settings-1 41
1.20 Settings-2 41
1.21 typeAliases 41
1.22 Environments-1 41
1.23 Environments-2 42
1.24 Environments-3 42
1.25 Environments-4 42
1.26 mappers 43
本章总结 43
本章作业 44
作业 1、查询用户表记录数 44
作业 2、实现用户表增删查改操作 44
学习任务 2：MyBatis 基础知识 45
本章目标 45
本章内容（学习活动） 45
1.1 SQL 映射的 XML 文件 45
1.2 Select-1 45
1.3 Select-2 46
1.4 Select-2 46
1.5 Insert 47
1.6 Update 47
1.7 Delete 48
1.8 演示示例——数据表操作 48
1.9 resultMap 49
1.10 动态 SQL 49
1.11 缓存 50
本章总结 50

本章作业……50
作业 1、使用 resultMap 实现数据库的操作……51
作业 2、使用 resultMap 实现关联数据的查询……51
作业 3、使用动态 SQL……51
学习任务 3：MyBatis-Spring 框架集成……52
本章目标……52
本章内容（学习活动）……52
1.1 基础知识小结-1……52
1.2 基础知识小结-2……52
1.3 MyBatis 工作流程……53
1.4 Spring 和 MyBatis 整合步骤……53
1.5 MyBatis 与 Spring 整合方式-1……54
1.6 MyBatis 与 Spring 整合方式-2……54
1.7 SpringMVC 与 MyBatis 整合框架……54
本章总结……55
本章作业……55

前言（课程导学）

一、课程安排

本课程将会讲解 SpringMVC+MyBatis 高级框架技术为主，包含的章节有：

第一章　：Spring MVC 框架

第二章　：MyBatis 框架

二、课程学习目标

- 学完本门课程后，你能够：
  - 了解 Spring　MVC 的架构以及请求流程
  - 搭建 Spring　MVC 环境
  - 完成 Controller 和 Viewer 的映射
  - 掌握异常处理
  - 使用 Servlet　API 对象作为入参
  - Spring　MVC 访问静态文件
  - 掌握文件上传处理
  - 掌握 Spring　MVC 返回 JSON 数据
  - 掌握 MyBatis 核心类的作用域以及生命周期
  - 熟练运用 MyBatis 框架进行开发
  - 掌握与 Spring 框架集成
  - 使用 SSM 或 SpringMVC+MyBatis 框架开发项目

三、前导课程

传统核心框架之 SSH 课程，内容主要有 Hibernate 查询、关联映射、缓存、Struts2 基础、配置、拦截器和 Spring、MVC、视图、缓存、文件上传等。

四、后继课程

主要的后继课程有：MyBatis、项目课等。

第一章 SpringMVC 框架

学习任务 1：Spring MVC 框架-1

回顾与作业点评

- 相关课程回顾
  - SSH 框架整合

本章任务

- 任务 1：搭建 Spring MVC 环境
- 任务 2：使用 Spring MVC 实现用户管理的增加、修改、删除、查询功能

本章目标

- 学完本次课程后，你能够：
  - 了解 Spring MVC 的架构以及请求流程
  - 搭建 Spring MVC 环境
  - 完成 Controller 和 Viewer 的映射
  - 参数传递
  - 了解 REST 风格
  - 掌握数据校验（JSR-303）

本章内容（学习活动）

1.1 Spring MVC 架构

- Spring MVC
  - 结构最清晰的 MVC Model2 实现
  - Controller

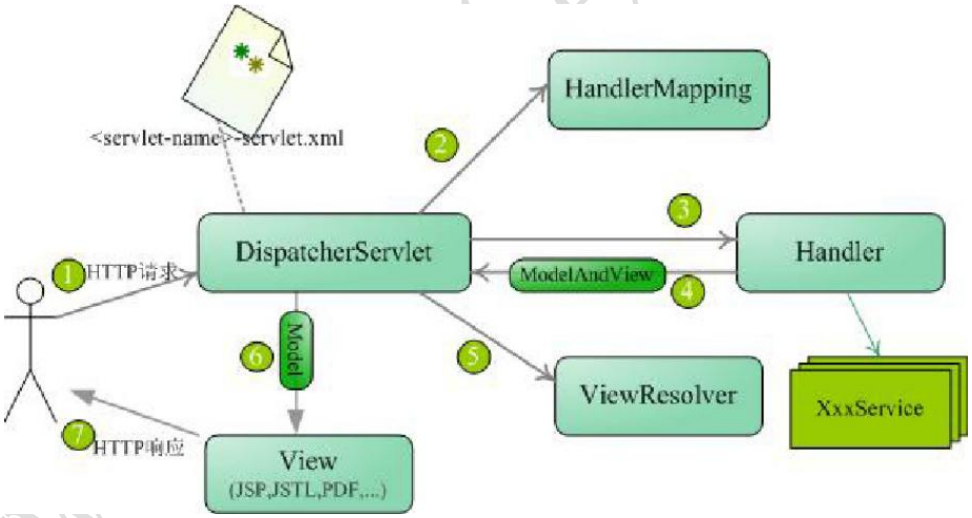
- ModelAndView

1.2 Spring MVC 处理请求流程

◆ 流程

- 用户发送请求前端控制器
- 前端控制器委托请求给处理器
- 页面控制器/处理器调用业务对象
- 模型返回模型数据
- 页面控制器/处理器返回 ModelAndView
- 前端控制器渲染视图
- 视图返回控制
- 前端控制器产生响应给用户

1.3 Spring MVC 框架模型



1.4 Spring MVC 特点

◆ Spring MVC 特点

- 清晰地角色划分
- 灵活的配置功能
- 提供了大量的控制器接口和实现类
- 真正的 View 层实现无关（JSP、Velocity、Xslt 等）
- 国际化支持

- 面向接口编程
- Spring 提供了 Web 应用开发的一整套流程，不仅仅是 MVC，他们之间可以很方便的结合一起

1.5 Hello,SpringMVC-1

- 需求说明
  - 创建 Spring MVC 实例，在 jsp 页面上输出“Hello，SpringMVC”
  - 使用 BeanNameUrlHandlerMapping 方式
- 分析
  - 创建工程并添加 jar 包到项目中
  - 编写配置文件以及代码
    - web.xml
    - dispatcherServlet-servlet.xml
- 演示示例：Hello,SpringMVC-1
  - 核心代码片段

```
public class IndexController extends AbstractController{

    @Override
    protected ModelAndView handleRequestInternal(HttpServletRequest arg0,
            HttpServletResponse arg1) throws Exception {
        // TODO Auto-generated method stub
        System.out.println("hello,SpringMVC!");
        return new ModelAndView("index");
    }


}
```

- 完成时间：15 分钟
- 教学方法：共性问题集中讲解

1.6 Hello,SpringMVC-2

- 需求说明
  - 创建 Spring MVC 实例，在 jsp 页面上输出“Hello，SpringMVC”

- 使用 DefaultAnnotationHandlerMapping 方式

◆ 分析

- 创建工程并添加 jar 包到项目中
- 编写配置文件以及代码
  - web.xml
  - dispatcherServlet-servlet.xml
- @Controller、@RequestMapping

◆ 演示示例：Hello,SpringMVC-2

- 核心代码片段

```
@Controller
public class IndexController{

    //RequestMapping表示用哪个url来对应
    @RequestMapping({"/index","/"})
    public String index(){
        System.out.println("hello,SpringMVC!");
        return "index";
    }

    @RequestMapping("/welcome")
    public String welcome(){
        return "welcome";
    }

}
```

◆ 完成时间：15 分钟

◆ 教学方法：共性问题集中讲解

1.7 HandlerMapping

◆ HandlerMapping

- BeanNameUrlHandlerMapping
  - 根据控制器 Bean 的名字将控制器映射到 URL
- ControllerBeanNameHandlerMapping
- ControllerClassNameHandlerMapping

- SimpleUrlHandlerMapping
- DefaultAnnotationHandlerMapping
  - 将请求映射给使用@RequestMapping 注解的控制器和控制方法

1.8 视图解析器

◆ 视图解析器
- 将逻辑视图的名字与 JSP 等视图技术进行匹配
- InternalResourceViewResolver
  - 在 Web 应用程序的 WAR 文件中查找视图模板，视图模板的路径根据加完前缀和后缀的逻辑视图名称来确定
  - prefix
  - suffix
- <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
  <property name="prefix" value="/WEB-INF/jsp/"/>
  <property name="suffix" value=".jsp"/>
  </bean>
  - /WEB-INF/jsp/index.jsp
    - /WEB-INF/jsp：前缀
    - Index：逻辑视图
    - .jsp：后缀

1.9 参数传递-1

◆ 需求说明
- 在上一个演示示例基础上进行参数的传递，后台打印出输入的 URL 参数值

◆ 分析
- @RequestParam
- 直接通过函数参数传值

◆ 演示示例：把值传给 Controller
- 核心代码片段

```
@Controller
public class IndexController{

    //RequestMapping表示用哪个url来对应
//  @RequestMapping({"/index","/"})
//  public String index(@RequestParam("username") String username){
//      System.out.println("hello,SpringMVC!");
//      System.out.println(username);
//      return "index";
//  }

    @RequestMapping({"/index","/"})
    public String index(String username){
        System.out.println("hello,SpringMVC!");
        System.out.println(username);
        return "index";
    }

    @RequestMapping("/welcome")
    public String welcome(){
        return "welcome";
    }

}
```

- 完成时间：10 分钟
- 教学方法：共性问题集中讲解

1.10 参数传递-2

- 需求说明
  - 在上一个演示示例基础上进行参数的传递，页面上显示参数的值
- 分析
  - Map<String,Object>
  - Model
- 演示示例：Controller 把值传给 View
  - 核心代码片段

```
@RequestMapping({"/index","/"})
public String index(String username,Model model){
    System.out.println("hello,SpringMVC!");
    model.addAttribute("username", username);
    //此时默认使用对象的类型作为key--->model.addAttribute("string", username)
    //model.addAttribute(new User())----> model.addAttribute("user", new User())
    model.addAttribute(username);
    System.out.println(username);
    return "index";
}
```

- 完成时间：10 分钟
- 教学方法：共性问题集中讲解

1.11 使用 JSTL 实现页面输出

- 需求说明
  - 在上一个演示示例基础上修改 index.jsp,使用 JSTL 的方式输出页面内容
- 分析
  - 引入 jstl.jar、standard.jar
  - 修改 index.jsp 代码
- 演示示例 ：使用 JSTL 标签实现页面输出
  - 核心代码片段

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>第一个Spring MVC实例</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
  </head>
```

```
  <body>
    <h1>hello,SpringMVC! <c:out value="${username}"/></h1>
    <h1><c:out value="${string}"/></h1>
  </body>
</html>
```

1.12 REST 风格

- REST 风格
  - Representational State Transfer 表述性状态转移
  - 传统的查、改、删的 URL 与 REST 风格的增删改 URL 对比
    - /userview.action?id=12 VS /user/12
    - /userdelete.action?id=12 VS /user/12/delete
    - /userupdate.action?id=12 VS /user/12/update
  - 请求方式
    - GET
    - POST
    - DELETE
    - PUT
  - 他强调的是一个资源可以对应多种视图

1.13 Spring MVC 实现用户管理-1

- 需求说明
  - 使用 Spring MVC 实现用户管理之用户列表展示
- 分析
  - 创建 POJO — User 类
  - HashMap
  - @RequestMapping(value="/userlist",method=RequestMethod.GET)
- 演示示例：SpringMVC 实现用户管理之用户列表
  - 核心代码片段

```
private Map<String, User> userList= new HashMap<String, User>();

public UserController() {
    userList.put("hl", new User("hl","123456","部门经理","hanlu@bdqn.cn"));
    userList.put("zs", new User("zs","123456","质量经理","zhangsan@bdqn.cn"));
    userList.put("zw", new User("zw","123456","开发工程师","zhangwei@bdqn.cn"));
    userList.put("ly", new User("ly","123456","实施顾问","liyu@bdqn.cn"));
}

@RequestMapping(value="/userlist",method=RequestMethod.GET)
public String list(Model model){
    model.addAttribute("userlist",userList);
    return "user/list";
}
```

- 完成时间：15 分钟
- 教学方法：共性问题集中讲解

1.14 Spring MVC 实现用户管理-2

- 需求说明
  - 使用 Spring MVC 实现用户管理之新增用户分析
- 分析
  - RequestMapping(value="/add",method=RequestMethod.POST)
  - <%@taglib prefix="sf"

    uri="http://www.springframework.org/tags/form"%>
  - 服务器端验证(JSR-303)
- 演示示例：SpringMVC 实现用户管理之增加用户

  核心代码片段

```
//链接到add页面时，为get请求，访问以下代码（两种方式）
@RequestMapping(value="/adduser",method=RequestMethod.GET)
/* 方式一：
 public String add(Model model){
   model.addAttribute(new User());
   return "user/add";
}
```

- 完成时间：20 分钟
- 教学方法：共性问题集中讲解

1.15 Spring MVC 实现用户管理-3

- 需求说明

- 使用 Spring MVC 实现用户管理之查看用户明细，并修改用户信息

◆ 分析

- @RequestMapping(value="/{username}/update",method=RequestMethod.POST)
- @PathVariable

◆ 演示示例：SpringMVC 实现用户管理之修改用户

- 核心代码片段

```
//修改用户信息,链接到update页面时，为get请求
@RequestMapping(value="/{username}/update",method=RequestMethod.GET)
public String update(@PathVariable String username,Model model){
    model.addAttribute(userList.get(username));
    return "user/update";
}
```

◆ 完成时间：20 分钟

◆ 教学方法：共性问题集中讲解

1.16 Spring MVC 实现用户管理-4

◆ 需求说明

- 使用 Spring MVC 实现用户管理之删除指定用户

◆ 分析

- @RequestMapping(value="/{username}/delete",method=RequestMethod.GET)
- @PathVariable

◆ 演示示例：SpringMVC 实现用户管理之删除用户

- 核心代码片段

```
//删除用户信息
@RequestMapping(value="/{username}/delete",method=RequestMethod.GET)
public String update(@PathVariable String username){
    userList.remove(username);
    return "redirect:/user/userlist";
}
```

◆ 完成时间：10 分钟

◆ 教学方法：共性问题集中讲解

本章总结

 Q：

- 简述 Spring MVC 的框架模型？
- Spring MVC 框架搭建有哪些步骤？
- 如何把值传递给 Controller ？
- Controller 如何把值传递给 Viewer？
- REST 风格
- Spring MVC 提供对 JSR-303 验证方式的支持

本章作业

根据课上讲解内容，完成演示示例和课堂练习

作业 1、Hello,SpringMVC-1

需求说明：

1) 创建 Spring MVC 实例，在 jsp 页面上输出“Hello，SpringMVC”
2) 使用 BeanNameUrlHandlerMapping 方式

分析：

1) 创建工程并添加 jar 包到项目中
2) 编写配置文件以及代码(web.xml、dispatcherServlet-servlet.xml)

作业 2、Hello,SpringMVC-2

需求说明：

1) 创建 Spring MVC 实例，在 jsp 页面上输出“Hello，SpringMVC”
2) 使用 DefaultAnnotationHandlerMapping 方式实现

分析：

1) 创建工程并添加 jar 包到项目中
2) 编写配置文件以及代码(web.xml、dispatcherServlet-servlet.xml)
3) @Controller、@RequestMapping

作业 3、参数传递-把值传给 Controller

需求说明：

1) 上练习 2 的基础上进行参数的传递，后台打印出输入的 URL 参数值

分析：

1) @RequestParam
2) 直接通过函数参数传值

作业 4、参数传递- Controller 把值传给 View

需求说明：

1) 上练习 3 的基础上进行参数的传递，页面上显示参数的值。

分析：

1) Map<String,Object>
2) Model

作业 5、使用 JSTL 标签实现页面输出

需求说明：

1) 在练习 4 的基础上修改 index.jsp,使用 JSTL 的方式输出页面内容

分析：

1) 引入 jstl.jar、standard.jar
2) 修改 index.jsp 代码

作业 6、SpringMVC 实现用户管理之用户列表

需求说明：

1) 使用 Spring MVC 实现用户管理之用户列表展示

分析：

1) 创建 POJO — User 类
2) HashMap
3) @RequestMapping(value="/userlist",method=RequestMethod.GET)

作业 7、SpringMVC 实现用户管理之增加用户

需求说明：

1) 使用 Spring MVC 实现用户管理之新增用户

分析：

1) RequestMapping(value="/add",method=RequestMethod.POST)
2) <%@taglib prefix="sf" uri="http://www.springframework.org/tags/form" %>
3) 服务器端验证(JSR-303)

作业 8、SpringMVC 实现用户管理之修改用户

需求说明：

1) 使用 Spring MVC 实现用户管理之查看用户明细，并修改用户信息

分析：

1) @RequestMapping(value="/{username}/update",method=RequestMethod.POST)
2) @PathVariable

作业 9、SpringMVC 实现用户管理之删除用户

需求说明：

1) 使用 Spring MVC 实现用户管理之删除指定用户

分析：

1) @RequestMapping(value="/{username}/delete",method=RequestMethod.GET)
2) @PathVariable

学习任务 2：Spring MVC 框架-2

回顾与作业点评

- Spring MVC 框架搭建
- 参数传递

本章任务

- 任务 1：使用 Spring MVC 实现用户管理的登录功能
- 任务 2：Spring MVC 访问静态文件
- 任务 3：处理文件上传
- 任务 4：返回 JSON 数据

本章目标

- 学完本次课程后，你能够：
  - 掌握异常处理
  - 使用 Servlet API 对象作为入参
  - Spring MVC 访问静态文件
  - 掌握文件上传处理
  - 掌握 Spring MVC 返回 JSON 数据

本章内容（学习活动）

1.1 Spring MVC 实现用户登录

- 需求说明
  - 使用 Spring MVC 实现用户管理之用户登录
- 分析
  - 异常处理
    - 局部异常处理：@ExceptionHandler
    - 全局异常处理：SimpleMappingExceptionResolver

- 使用 Servlet API 对象作为入参
  - HttpSession
  - HttpServletRequest
  - HttpServletResponse

◆ 演示示例：SpringMVC 实现用户管理之用户登录

- 课上演示

◆ 完成时间：15 分钟

◆ 教学方法：共性问题集中讲解

1.2 静态资源文件处理

◆ 需求说明

- 在上一个演示示例的基础上，为其增加样式

◆ 分析

- <mvc:resources/>

◆ 演示示例 ：静态资源文件的引用

```
<!-- 将静态文件指定到某个特殊的文件夹中统一处理 -->
<mvc:resources mapping="/resources/**" location="/resources/" />
<!-- 对转向页面的路径解析。prefix：前缀， suffix：后缀 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
<property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

◆ 完成时间：10 分钟

◆ 教学方法：共性问题集中讲解

1.3 处理文件上传-单文件上传

◆ 需求说明

- 在用户管理的添加页面，上传附件

◆ 分析

- 导包
  - commons-fileupload-1.2.2.jar、commons-io-2.4.jar

- 在添加页面表单中添加一个文件上传域
- 修改 UserController 的 add()方法以接收上传的文件
- 在 Spring 中配置 multipart 文件处理器

◆ 演示示例 ：处理文件上传-单文件上传

- 核心代码片段

```
//添加用户时，为post请求，访问以下代码
@RequestMapping(value="/adduser",method=RequestMethod.POST)
//紧跟validate之后写验证结果
public String add(@Validated User user,BindingResult bindingResult,MultipartFile attach,HttpServletRequest req){
    if(bindingResult.hasErrors()){
        //若有错误，直接掉转到add 视图
        return "user/add";
    }
    //判断文件是否为空
    if(!attach.isEmpty()){
        String realPath =  req.getSession().getServletContext().getRealPath("/resources/upload");
        System.out.println(realPath);
        File file = new File(realPath+"/"+attach.getOriginalFilename());

        try {
            FileUtils.copyInputStreamToFile(attach.getInputStream(), file);
            //FileUtils.writeByteArrayToFile(file, attach.getBytes());
            //attach.transferTo(file);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println(attach.getName()+" ======= "+attach.getOriginalFilename()+" ====== "+attach.getContentType());
    }
    userList.put(user.getUserName(), user);
    return "redirect:/user/userlist";
}
```

◆ 完成时间：20 分钟

◆ 教学方法：共性问题集中讲解

1.4 处理文件上传-多文件上传

◆ 需求说明

- 在上一示例的基础上，实现多文件上传

◆ 分析

- 修改 UserController 的 add()方法，将 MultipartFile 参数类定义成数组

- 演示示例 ：处理文件上传-多文件上传
  - 核心代码片段

```
//添加用户时，为post请求，访问以下代码
@RequestMapping(value="/adduser",method=RequestMethod.POST)
//紧跟validate之后写验证结果
public String add(@Validated User user,BindingResult bindingResult,
@RequestParam("attachs") MultipartFile[] attachs,HttpServletRequest req){
    if(bindingResult.hasErrors()){
        //若有错误，直接掉转到add 视图
        return "user/add";
    }
    String realPath =  req.getSession().getServletContext().getRealPath("/resources/upload");
    System.out.println(realPath);

    for(MultipartFile attach:attachs){
        if(!attach.isEmpty()){//判断文件是否为空
            File file = new File(realPath+"/"+attach.getOriginalFilename());
            try {
                //FileUtils.copyInputStreamToFile(attach.getInputStream(), file);
                //FileUtils.writeByteArrayToFile(file, attach.getBytes());
                attach.transferTo(file);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    userList.put(user.getUserName(), user);
    return "redirect:/user/userlist";
}
```

- 完成时间：20 分钟
- 教学方法：共性问题集中讲解

1.5 Spring MVC 返回 JSON 数据

- 需求说明
  - 修改 view 页面，在页面上显示返回的 JSON 数据
- 分析
  - 导入 jar 包：jackson-all-1.9.11.jar
  - @RequestMapping(value="/{username}",method=RequestMethod.GET,params="json")
  - @ResponseBody
- 演示示例 ：Spring MVC 返回 JSON 数据
  - 核心代码片段

```
//查看用户信息 ?json
@RequestMapping(value="/{username}",method=RequestMethod.GET,params="json")
@ResponseBody
public User view(@PathVariable String username){
    return userList.get(username);
}
```

- 完成时间：10 分钟
- 教学方法：共性问题集中讲解

本章总结

- 异常处理
- 使用 Servlet API 对象作为入参
- <mvc:resources/>
- 文件上传
- 返回 JSON 数据
- DispatcherServlet
- Handler
  - @Controller
  - @RequestMapping（value、method、params）
  - @PathVariable
  - @RequestParam
  - @ResponseBody
- 方法的返回值
  - ModelAndView
  - View
  - Model
  - String
  - Object

- Void

本章作业

根据课上讲解内容，完成演示示例和课堂练习

作业 1、SpringMVC 实现用户管理之用户登录

需求说明：

1) 使用 Spring MVC 实现用户管理之用户登录

分析：

1) 异常处理（局部异常处理：@ExceptionHandler、全局异常处理：SimpleMappingExceptionResolver）
2) 使用 Servlet API 对象作为入参(HttpSession、HttpServletRequest、HttpServletResponse)

作业 2、静态资源文件的引用

需求说明：

1) 在练习 1 的基础上，为其增加样式
2) 使用 DefaultAnnotationHandlerMapping 方式实现

分析：

1) <mvc:resources/>

作业 3、处理文件上传-单文件上传

需求说明：

1) 在用户管理的添加页面，上传附件

分析：

1) 导包：commons-fileupload-1.2.2.jar、commons-io-2.4.jar
2) 在添加页面表单中添加一个文件上传域
3) 修改 UserController 的 add()方法以接收上传的文件
4) 在 Spring 中配置 multipart 文件处理器

作业 4、处理文件上传-多文件上传

需求说明：

1) 上练习 3 的基础上实现多文件上传。

分析：

1) 修改 UserController 的 add()方法，将 MultipartFile 参数类定义成数组

作业 5、Spring MVC 返回 JSON 数据

需求说明：

1) 修改 view 页面，在页面上显示返回的 JSON 数据

分析：

1) 导入 jar 包：jackson-all-1.9.11.jar
2) @RequestMapping(value="/{username}",method=RequestMethod.GET,params="json")
3) @ResponseBody

第二章 MyBatis 框架

学习任务 1：初识 MyBatis

本章目标

- 学完本次课程后，你能够：
  - 理解 MyBatis 的概念以及优点特性
  - 理解核心类的作用域和生命周期
  - 了解 MyBatis 与 JDBC、Hibernate 的区别
  - 掌握配置文件结构内容
  - 搭建 MyBatis 环境

本章内容（学习活动）

1.1 MyBatis 简介

- MyBatis 的前身是 iBatis，本是 Apache 的一个开源的项目
- MyBatis 是一个数据持久层(ORM)框架,把实体类和 SQL 语句之间建立了映射关系 ,是一种半自动化的 ORM 实现。
- MyBatis 小巧，简单易学
  - 基于 SQL 语法，简单易学
  - 能了解底层组装过程
  - SQL 语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度
  - 程序调试方便

1.2 搭建 MyBatis 开发环境

- 使用 MyBatis 的开发步骤：
  - 下载需要的 MyBatis jar 包

- 部署 MyBatisjar 包
- 编写 MyBatis 配置文件
- 创建实体类和 DAO 接口
- 创建 SQL 映射文件
- 创建 DAO 接口的实现类
- 编写测试类进行测试

1.3 查询用户表记录数

◆ 需求说明

- 搭建 MyBatis 开发环境，实现用户表记录数的查询
  - 在 MyEclipse 中创建工程，导入 MyBatis 的 jar 包
  - 创建 MyBatis 配置文件 mybatis-config.xml，配置数据库信息
  - 编写实体类 User 和 UserMapper 接口
  - 配置映射文件 UserMapper.xml
  - 实现 UserImpl

◆ 演示示例 1：用户表记录数查询

核心代码片段

```
<mapper namespace="cn.bdqn.dao.UserMapper">

    <select id="count" resultType="int">
        select count(1) from user
    </select>


</mapper>
```

1.4 与 JDBC 直观对比

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection conn = DriverManager.getConnection(url, username, password);
java.sql.PreparedStatement st = conn.prepareStatement(sql);
st.setInt(0, 1);
st.execute();
java.sql.ResultSet rs = st.getResultSet();
while(rs.next()){
   String resultString = rs.getString(columnname);
}
/**================华丽丽滴分割线============================**/
<mapper namespace="cn.jbit.dao.UserMapper">
    <select id="getUserList" resultType="user" parameterType="user">
        select * from user
    </select>
</mapper>
```

- MyBatis 将上面这几行代码分解包装：
  - 1、2 行：是对数据库的数据源的管理包括事务管理
  - 3、4 两行：MyBatis 通过配置文件来管理 SQL 以及输入参数的映射
  - 6、7、8 行：MyBatis 获取返回结果到 Java 对象的映射，也是通过配置文件管理

1.5 与传统 JDBC 的比较

- 减少了 61%的代码量
- 最简单的持久化框架
- 架构级性能增强
- SQL 代码从程序代码中彻底分离，可重用
- 增强了项目中的分工
- 增强了移植性

1.6 与 Hibernate 的比较

MyBatis	Hibernate
是一个 SQL 语句映射的框架（工具）	主流的 ORM 框架、提供了从 POJO 到数据库表的全套映射机制
注重 POJO 与 SQL 之间的映射关系。不会为程序员在运行期自动生成 SQL	会自动生成全套 SQL 语句
自动化程度低、手工映射 SQL,灵活程度高	因为自动化程度高、映射配置复杂，API 也相对复杂，灵活性低
需要开发人员熟炼掌据 SQL 语句	开发人员不必关注 SQL 底层语句开发

1.7 MyBatis 基本要素

- MyBatis 的核心对象
- configuration.xml 全局配置文件
- mapper.xml 核心映射文件

1.8 核心接口和类的结构

- SqlSessionFactoryBuilder

- build()

- SqlSessionFactory
  - openSession()
- SqlSession

1.9 核心对象

- SqlSessionFactoryBuilder
- SqlSessionFactory
- SqlSession
- 注意：理解 MyBatis 核心对象的不同范围和生命周期是很重要的。 不正确的使用它们会导 致严重的并发问题。

1.10 SqlSessionFactoryBuilder-1

- SqlSessionFactoryBuilder
  - 用过即丢，其生命周期只存在于方法体内
  - 可重用 SqlSessionFactoryBuilder 来创建多个 SqlSessionFactory 实例

1.11 SqlSessionFactoryBuilder-2

- SqlSessionFactoryBuilder
  - SqlSessionFactoryBuilder 类负责构建
  - SqlSessionFactory,并且提供了多个 build 的重载方法
    - public SqlSessionFactory build(InputStream inputStream, String environment,Properties properties)
    - public SqlSessionFactory build(Reader reader, String environment, Properties properties)
    - public SqlSessionFactory build(Configuration config)
      - 配置信息以三种形式提供给 SqlSessionFactory 的 build 方法：InputStream(字节流）、Reader（字符流）、Configuration（类）

1.12 SqlSessionFactoryBuilder-2

- SqlSessionFactoryBuilder
  - SqlSessionFactoryBuilder 类负责构建 SqlSessionFactory,并且提供了多个 build 的重载方法
  - 构建一个 SqlSessionFactory 的两种方式：
    - 读取 XML 文件构造方式
      - 读取 XML 文件构造方式：

        String resource = "mybatis-config.xml";

        InputStream is= Resources.getResourceAsStream(resource);

        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is) ;
    - 编程构造方式

1.13 SqlSessionFactory

- SqlSessionFactory
  - SqlSessionFactory 是每个 MyBatis 应用的核心
  - 单例，存在于整合应用运行时，并且同时只存在一个对象实例

1.14 SqlSession-1

- SqlSession
  - 包含了执行 sql 所需的所有方法，可以通过 SqlSession 实例直接运行映射的 sql 语句
  - SqlSession 对应着一次数据库会话，它的生命周期不是永久的，在每次访问数据库时都需要创建它
  - 每个线程都有自己的 SqlSession 实例，SqlSession 实例是不能被共享，也不是线程安全的

1.15 SqlSession-2

- 注意

1. 并不是说在 SqlSession 里只能执行一次 sql，你可以执行多次，当一旦关闭了 SqlSession 就需要重新创建它
2. 关闭 SqlSession 是非常重要，你必须要确保 SqlSession 在 finally 方法体中正常关闭

- SqlSessionFactory 的 openSession 方法：
  - SqlSession session = sqlSessionFactory.openSession();

    try {

        // do work

    } finally {

        session.close();

    }

1.16 SqlSession-3

- SqlSession 的获取与使用
  - SqlSession 的获取方式
    - String resource = "mybatis-config.xml";

      InputStream is = Resources.getResourceAsStream(resource);

      SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(is) ;

      SqlSession sqlSession = factory.openSession();
  - Sqlsession 的使用：
    - 调用 insert,update,selectList,selectOne,delete 等方法执行增、删、查、改等操作
- 演示示例 2：实现用户表增删查改操作

  核心代码片段

```
<select id="count" resultType="int">
    select count(1) from user
</select>

<insert id="add" parameterType="User">
    insert into user (userCode,userName,userPassword)
        values (#{userCode},#{userName},#{userPassword})
</insert>

<update id="update" parameterType="User">
    update user set userCode=#{userCode},userName=#{userName},
        userPassword=#{userPassword} where id=#{id}
</update>

<delete id="delete" parameterType="User">
    delete from user where id=#{id}
</delete>

<select id="getUserList" resultType="User">
    select * from user
</select>
```

1.17 configuration.xml

- configuration.xml-系统核心配置文件
  - 包含数据源和事务管理器等设置和属性信息
    - configuration 配置
      - properties 可以配置在 Java 属性配置文件中

        settings 修改 MyBatis 在运行时的行为方式

        typeAliases 为 Java 类型命名一个别名（简称）

        typeHandlers 类型处理器

        objectFactory 对象工厂

        plugins 插件

        environments 环境

        environment 环境变量

        transactionManager 事务管理器

        dataSource 数据源

1.18 properties

- properties 元素
  - properties 和 java 的.properties 的配置文件有关。配置 properties 的 resource 指定. 文件中相应属性值。properties 的路径，然后再在 properties 标签下配置 property 的 name 和 value，则可以替换.properties
    - <!– 属性替换-->

```
<properties resource= “jdbc.properties">
    <property name="driver" value="${driver}"/>
    <property name="url" value="${url}"/>
    <property name="username" value="${username}"/>
    <property name="password" value="${password}"/>
</properties>
```

1.19 Settings-1

- Settings 元素
  - 用来修改 MyBatis 在运行时的行为方式，主要是 Mybatis 的一些全局配置属性的设置

设置项	描述	允许值	默认值
cacheEnabled	对在此配置文件下的所有 cache 进行全局性开/关设置	true|false	true
lazyLoadingEnabled	全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。	true|false	true
……(9 个)	… …	… …	… …

1.20 Settings-2

- Settings 元素
  - 用来修改 MyBatis 在运行时的行为方式，主要是 Mybatis 的一些全局配置属性的设置
    - <settings>

```
<!--启用延迟加载-->
<setting name="lazyLoadingEnabled" value="false" />
</settings>
```

1.21 typeAliases

- typeAliases 元素
  - 类型别名是 Java 类型的简称,它仅仅只是关联到 XML 配置，简写冗长的 JAVA 类名。
    - <typeAliases>

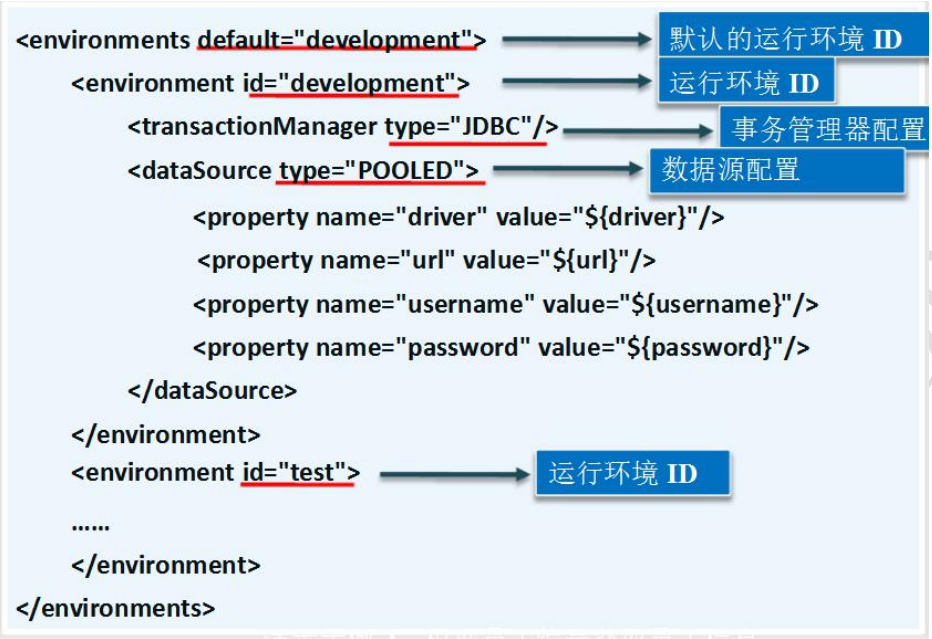
```
<typeAlias alias="User" type="cn.jbit.pojo.User"/>
</typeAliases>
```

1.22 Environments-1

- Environments 元素
  - MyBatis 可以配置多套运行环境，将 SQL 映射到多个数据库上
  - 虽然可以配置多个运行环境，但是每个 SqlSessionFactory 实例只能选择一个运行环境。即：每

个数据库对应一个 SqlSessionFactory 实例

1.23 Environments-2



1.24 Environments-3

- transactionManager-事务管理器
  - <transactionManager type= "[JDBC|MANAGED]"/>
    - JDBC
    - MANAGED（托管）

1.25 Environments-4

- dataSource
  - dataSource 元素使用基本的 JDBC 数据源接口来配置 JDBC 连接对象的资源。有三种内建的数据源类型
  - <dataSource type= "[UNPOOLED|POOLED|JNDI]"/>
    - UNPOOLED
    - POOLED
    - JNDI
  - 配置
    - <dataSource type="POOLED">

```
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
    </dataSource>
```

1.26 mappers

- mappers 元素
  - Sql 映射语句一般定义在各持久类的 Mapper.xml 文件中，需要在配置中引用这些映射文件
    - <!-- 将 mapper 文件加入到配置文件中-->

      ```
      <mappers>
        <mapper resource="cn/jbit/dao/UserMapper.xml"/>
      </mappers>
      ```

本章总结

- MyBatis 是支持普通 SQL 查询，存储过程和高级映射的优秀持久层框架
- MyBatis 支持
  - SQL 查询
  - 存储过程
  - 高级查询
- MyBatis 消除
  - 手工 jdbc 代码
  - 参数手工设置
  - 结果集的检索
- MyBatis 使用
  - Xml
  - 注解
    - pojo 映射成数据库的记录
- MyBatis 是支持普通 SQL 查询，存储过程和高级映射的优秀持久层框架

- MyBatis 环境搭建，实现对数据表简单的增删查改操作
- MyBatis 的核心对象
  - SqlSessionFactoryBuilder
  - SqlSessionFactory
  - SqlSession
- configuration.xml 全局配置文件

本章作业

根据课上讲解内容，完成演示示例和课堂练习

作业 1、查询用户表记录数

需求说明：

1) 搭建 MyBatis 开发环境，实现用户表记录数的查询

分析：

1) 在 MyEclipse 中创建工程，导入 MyBatis 的 jar 包
2) 创建 MyBatis 配置文件 mybatis-config.xml，配置数据库信息
3) 编写实体类 User 和 UserMapper 接口
4) 配置映射文件 UserMapper.xml
5) 实现 UserImp

作业 2、实现用户表增删查改操作

需求说明：

1) 在上一个演示示例的基础上，继续完成用户表的增删改查操作

分析：

1) 调用 insert,update,selectList,selectOne,delete 等方法执行增、删、查、改等操作

学习任务 2：MyBatis 基础知识

本章目标

- 学完本次课程后，你能够：
  - 掌握通过 Sql 映射 XML 文件进行增删改查
  - 掌握参数的使用
  - 掌握动态 sql 语句
  - 掌握 resultMap
  - 了解 Cache 的使用

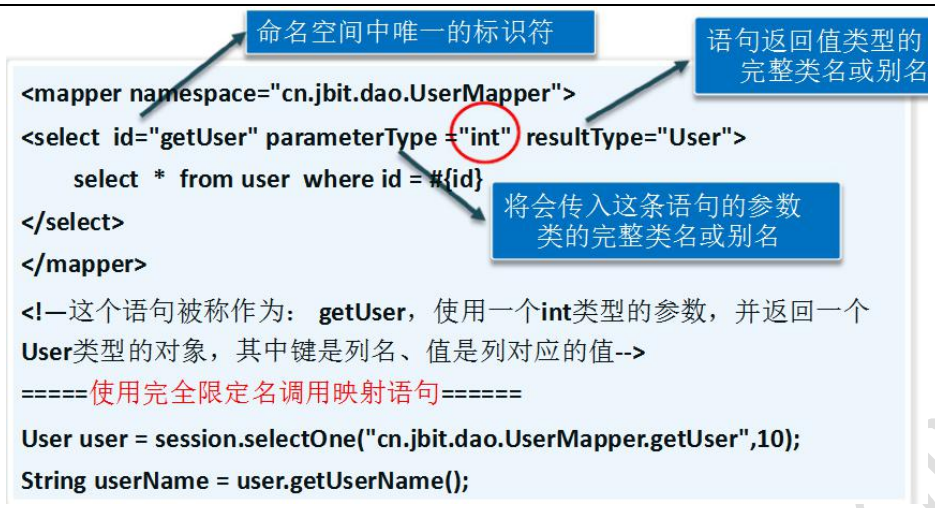
本章内容（学习活动）

1.1 SQL 映射的 XML 文件

- MyBatis 真正的力量是在映射语句中，专注于 SQL，功能强大，SQL 映射的配置却是相当简单
- SQL 映射文件的几个顶级元素(按照定义的顺序)
  - cache - 配置给定命名空间的缓存
  - cache-ref – 从其他命名空间引用缓存配置
  - resultMap –用来描述数据库结果集和对象的对应关系
  - sql – 可以重用的 SQL 块，也可以被其他语句引用
  - insert – 映射插入语句
  - update – 映射更新语句
  - delete – 映射删除语句
  - select – 映射查询语句

1.2 Select-1

- Select 是 MyBatis 中最常用的元素之一
- Select 语句有很多属性可以详细配置每一条语句

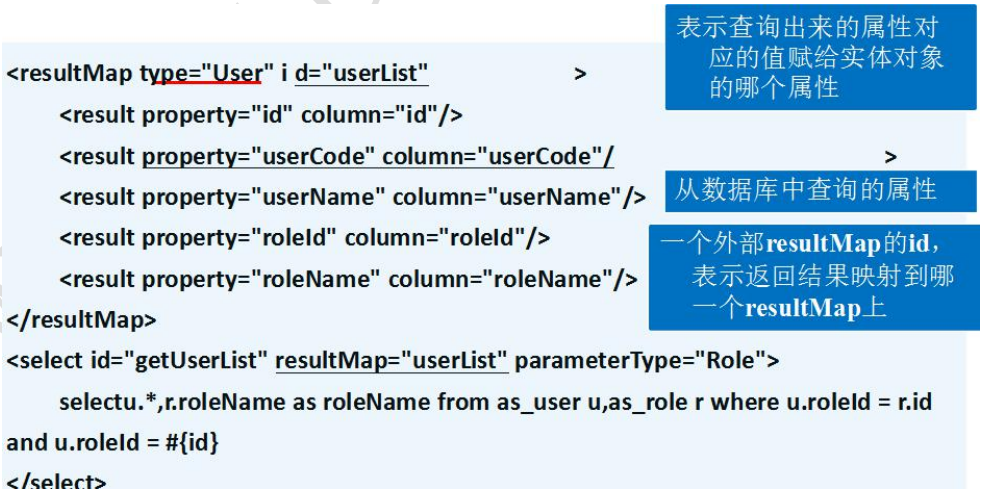



```
<mapper namespace="cn.jbit.dao.UserMapper">
<select id="getUser" parameterType ="int" resultType="User">
    select * from user where id = #{id}
</select>
</mapper>
<!—这个语句被称作为： getUser，使用一个int类型的参数，并返回一个User类型的对象，其中键是列名、值是列对应的值-->
=====使用完全限定名调用映射语句======
User user = session.selectOne("cn.jbit.dao.UserMapper.getUser",10);
String userName = user.getUserName();
```

```
<select id="getUser" parameterType ="User" resultType="User">
    select * from user where id = #{id}
</select>
=====使用完全限定名调用映射语句======
User user = session.selectOne("cn.jbit.dao.UserMapper.getUser",_user);
String userName = user.getUserName();
```

1.3 Select-2

- resultType ：直接表示返回类型
- resultMap ：对外部 resultMap 的引用
- 二者不能同时存在




```
<resultMap type="User" i d="userList"                    >
    <result property="id" column="id"/>
    <result property="userCode" column="userCode"/                    >
    <result property="userName" column="userName"/>
    <result property="roleId" column="roleId"/>
    <result property="roleName" column="roleName"/>
</resultMap>
<select id="getUserList" resultMap="userList" parameterType="Role">
    selectu.*,r.roleName as roleName from as_user u,as_role r where u.roleId = r.id
and u.roleId = #{id}
</select>
```

1.4 Select-2

属性	描述
id	在命名空间中唯一的标识符，可以被用来引用这条语句。

parameterType	将会传入这条语句的参数类的完全限定名或别名。
resultType	从这条语句中返回的期望类型的类的完全限定名或别名。注意集合情形，那应该是集合可以包含的类型，而不能是集合本身。使用 resultType 或 resultMap，但不能同时使用。
resultMap	命名引用外部的 resultMap
flushCache	将其设置为 true，不论语句什么时候被调用，都会导致缓存被清空。默认值：false。
useCache	将其设置为 true，将会导致本条语句的结果被缓存。默认值：true。
timeout	这个设置驱动程序等待数据库返回请求结果，并抛出异常时间的最大等待值。默认不设置（驱动自行处理）。
fetchSize	这是暗示驱动程序每次批量返回的结果行数。
statementType	STATEMENT,PREPARED 或 CALLABLE 的一种。让 MyBatis 选择使用 Statement，PreparedStatement 或 CallableStatement。默认值：PREPARED。
resultSetType	FORWARD_ONLY|SCROLL_SENSITIVE|SCROLL_INSENSITIVE 中的一种。默认是不设置（驱动自行处理）

1.5 Insert

- Insert
  - <insert id="addUser" parameterType="User">
    insert into user (userCode,userName,userPassword)values (
    #{userCode},#{userName},#{userPassword})
    </insert>

1.6 Update

- Update
  - <update id ="modifyUser" parameterType="User">

update user userCode = #{userCode},

userName = #{userName},

userPassword= #{userPassword} where id = #{id}

</update>

1.7 Delete

- Delete
  - <delete id ="deleteUser" parameterType="User">

    delete from user where id = #{id}

    </delete>

1.8 演示示例——数据表操作

- 需求说明
  - 用户表与角色表的关联关系（多对一）
  - 获取指定角色下的用户表数据列表（resultMap）
  - 完成对角色表的增、删、改操作
- 演示示例 1：使用 resultMap 实现数据库的操作

  核心代码片段

```
<!-- 当数据库中的字段信息与对象的属性不一致时需要通过resultMap来映射 -->
<resultMap type="User" id="seachUserResult">
    <result property="id" column="id"/>
    <result property="userCode" column="userCode"/>
    <result property="userName" column="userName"/>
    <result property="roleId" column="roleId"/>
    <result property="roleName" column="roleName"/>
</resultMap>


<select id="count" resultType="int">
    select count(1) from user
</select>

<insert id="add" parameterType="User">
    insert into user (userCode,userName,userPassword)
        values (#{userCode},#{userName},#{userPassword})
</insert>
```

1.9 resultMap

- resultMap - 描述如何将结果集映射到 Java 对象
- resultMap 属性
  - id
  - Type
- resultMap 子元素
  - id
  - result
  - association
  - collection
- 演示示例 2：使用 resultMap 实现关联数据的查询

  核心代码片段

```
<!-- 根据roleId获取用户列表 association start-->
<resultMap type="User" id="seachUserResult">
    <result property="id" column="id"/>
    <result property="userCode" column="userCode" />
    <result property="userName" column="userName" />
    <result property="roleId" column="roleId" />
    <!-- <association property="role" javaType="Role" >
        <result property="id" column="id"/>
        <result property="roleCode" column="roleCode"/>
        <result property="roleName" column="roleName"/>
    </association> -->
    <association property="role" javaType="Role" resultMap="roleMap"/>
</resultMap>
```

1.10 动态 SQL

- MyBatis 最强大的特性之一就是它的动态语句功能，使用动态 SQL 完成多条件查询
- 用于实现动态 SQL 的元素主要有
  - if
  - choose、when、otherwise
  - trim、where、set
  - foreach

1.11 缓存

- MyBatis 缓存
  - 一级缓存
  - 二级缓存
- 二级缓存的配置
  - MyBatis 的全局 cache 配置
    - <settings>
      <setting name="cacheEnabled" value="true"/>
      </settings>
  - 在 Mapper XML 文件中设置缓存，默认情况下是没有开启缓存的
    - <cache eviction="FIFO" flushInterval="60000"
      size="512" readOnly="true"/>
  - 在 Mapper XML 文件配置支持 cache 后，如果需要对个别查询进行调整，可以单独设置 cache
    - <select id="selectAll" resultType="Emp"
      useCache="true">

本章总结

- Sql 映射 XML 文件进行增删改查操作
- resultMap
- 动态 sql 语句
  - if
  - choose、when、otherwise
  - trim、where、set
  - foreach
- 缓存

本章作业

根据课上讲解内容，完成演示示例和课堂练习

作业 1、使用 resultMap 实现数据库的操作

需求说明：

1) 用户表与角色表的关联关系（多对一）
2) 获取指定角色下的用户表数据列表（resultMap）
3) 完成对角色表的增、删、改操作

作业 2、使用 resultMap 实现关联数据的查询

需求说明：

1) 在上一个演示示例的基础上，根据 roleId 获取用户列表
2) 获取指定用户的地址列表(user 表-address 表：1 对多关系)

分析：

1) Association
2) collection

作业 3、使用动态 SQL

需求说明：

1) 在上一个演示示例的基础上，根据条件，获取用户表数据列表(动态 sql)
2) 根据部门条件，获取用户表数据列表-foreach

分析：

1) If
2) choose、when、otherwise
3) trim、where、set

foreach

学习任务 3：MyBatis-Spring 框架集成

本章目标

- 学完本次课程后，你能够：
  - 理解 MyBatis 的工作流程原理
  - 掌握 Spring 集成 MyBatis
  - 掌握 SpringMVC+MyBatis 框架的搭建

本章内容（学习活动）

1.1 基础知识小结-1

- 优秀的 ORM 框架
- 核心类
  - SqlSessionFactory
  - SqlSessionFactoryBuilder
  - SqlSession
- configuration.xml 全局配置文件
  - properties
  - settings
  - typeAliases
  - environments
  - mappers

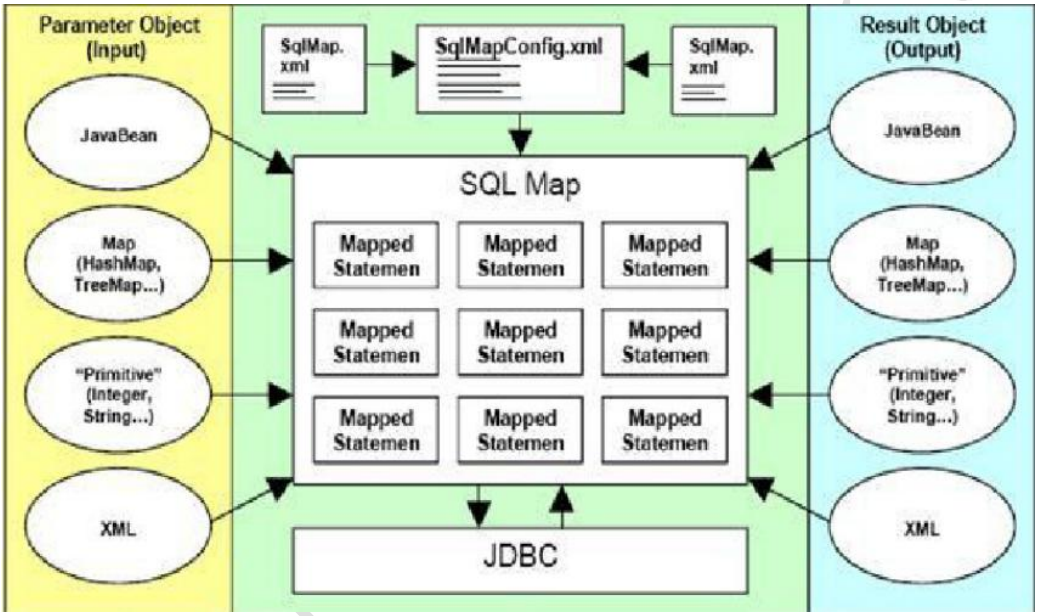
1.2 基础知识小结-2

- SQL 映射 Mapper.xml 文件
  - 增删查改操作
    - insert
    - update
    - delete

- select

- resultMap
- 动态 sql 语句
  - if
  - choose、when、otherwise
  - trim、where、set
  - foreach

1.3 MyBatis 工作流程



1.4 Spring 和 MyBatis 整合步骤

- 步骤
  - 建立工程，加入 Spring 和 MyBatis 的有关 Jar
  - 建立开发目录结构，创建实体类
  - 创建数据访问接口
  - 创建数据访问接口的实现类
  - 配置 SQL 映射语句文件
  - 配置 MyBatis 应用配置文件
  - 配置 Spring 应用配置文件

1.5 MyBatis 与 Spring 整合方式-1

- 使用 SqlSessionTemplate 完成数据库的操作
  - 实现员工的增加功能
  - 实现员工的修改功能
  - 实现员工的删除功能
  - 实现查询全部员工的功能
- 演示示例 1：使用 SqlSessionTemplate 实现数据库的操作

  核心代码片段

```
private SqlSessionTemplate sqlSessionTemplate;

protected ApplicationContext ctx = null;
private EmployeeDaoTest test;

private Logger logger = Logger.getLogger(EmployeeDaoTest.class);

@Before
public void setUp() throws Exception{
    ctx = new ClassPathXmlApplicationContext("applicationContext-mybatis.xml");
    test = (EmployeeDaoTest)ctx.getBean("employeeDaoTest");
}

@Test
public void getEmployeeListTest() {
    List<Employee>  list = test.sqlSessionTemplate.selectList("cn.bdqn.dao.EmployeeMapper.getEmployeeList");
    logger.debug("testGetEmployeeList---> " + list.size());
}
```

1.6 MyBatis 与 Spring 整合方式-2

- 采用数据映射器（MapperFactoryBean）的方式，完成对数据库的操作
  - 实现根据部门查询出该部门下的员工信息
  - 实现查询员工详细信息的功能
- 演示示例 2：使用映射接口实现数据库的操作

  课上演示

1.7 SpringMVC 与 MyBatis 整合框架

- 导入相关 jar 包
- 配置文件(/resources)
  - applicationContext-mybatis.xml

- jdbc.properties
- spring-servlet.xml
- mybatis-config.xml
- log4j.properties

- Dao 数据访问接口(cn.jbit.dao)
- 系统服务接口对象(cn.jbit.service)
- 数据对象模型(cn.jbit.pojo)
- Controller(cn.jbit.controller)
- JSP 页面(/WEB-INF/pages)
- 演示示例 3：搭建 SpringMVC+Mybatis 框架并实现数据库的操作

  课上演示

本章总结

- Spring 与 Mybatis 集成的两种方式
  - SqlSessionTemplate
  - 映射接口
- SpringMVC+MyBaits 框架搭建
- 课后作业
  - 完成课堂演示 demo
- 预习作业
  - 预习下节课内容

本章作业

把老师授课的 3 个案例熟练的再操作一遍。