课程目标

- 理解 MyBatis 逆向工程能解决什么问题，掌握怎么用。
- 理解为什么要 SSM 集成，SSM 集成本质是什么，利用 Spring 什么功能做什么。
- 掌握 Spring MVC + Spring + MyBatis 集成。

一、练习目标（理解）

1、需求

完成部门基本的 CRUD 和分页查询，完成员工基本的 CRUD、分页查询和过滤查询（根据姓名和邮箱模糊查询，根据部门查询）。

2、技术架构

使用 Spring MVC + Spring + MyBatis，数据库选用 MySQL，视图选用 JSP。

3、SSM 集成作用及本质

作用：在框架上基础上开发，发挥各个框架在各层的好处，提高开发效率。

本质：

- Spring 去集成 Spring MVC 和 MyBatis，即控制器对象、业务对象、Mapper 对象等都交由 Spring 容器管理，使用 Spring IoC 和 DI 来完成对象创建及其属性注入；
- 使用 AOP 来配置事务；
- 使用 Spring MVC 解决 MVC 的问题，处理请求和响应。

4、集成步骤

- 先用 Spring 集成 MyBatis
    - 搭建项目，添加依赖配置插件。
    - 把 Spring 和 MyBatis 的等配置文件拷贝进项目 resources 目录下
    - 配置数据库链接池
    - 配置 SqlSessionFactory
    - 配置 Mapper 对象
    - 配置业务对象
    - 配置事务相关
- 再加入 Spring MVC
    - 在 web.xml 配置端控制器和编码过滤器

- 在项目 resources 目录下添加 Spring MVC 的配置文件，并配置 MVC 注解解析器，扫描控制器，静态资源处理，视图解析器等等
- 在 Spring MVC 的配置文件中引入 Spring 配置文件

5、项目准备

5.1 在之前项目上改

这里所谓之前的项目就是 ssm 项目。

5.2、检查依赖和插件

确定依赖和插件都没有配错，或者少配。

二、MyBatis 逆向工程（掌握）

1、MyBatis 逆向工程

一个 Maven 插件，根据数据的表生成实体类和 Mapper 接口和 Mapper XML。

2、插件使用

2.1、新建表

参考文档，在 ssm 库新建如下表：

```
CREATE TABLE `department` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `sn` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2.2、配置插件

在 pom.xml，配置 MyBatis 逆向工程插件：

```
<!-- MyBatis 逆向工程插件 -->
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.3.2</version>
    <configuration>
        <verbose>true</verbose>
        <overwrite>false</overwrite>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.45</version>
        </dependency>
    </dependencies>
```

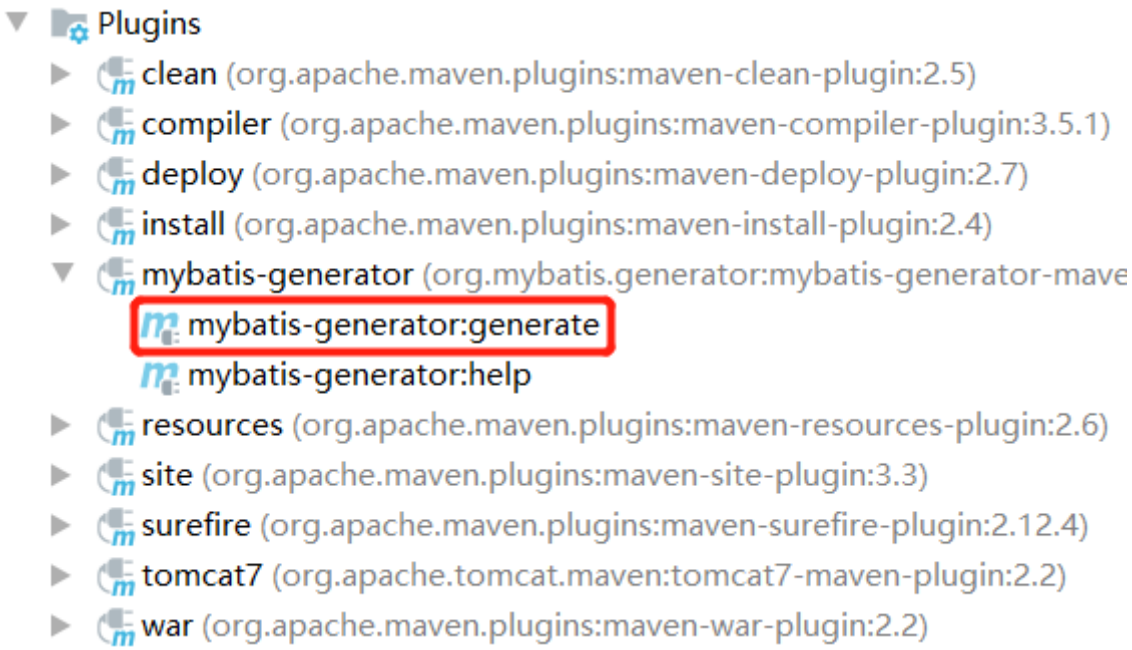
```
</plugin>
```

2.3、插件设置

在项目中 **resources 目录**中新建配置文件 generatorConfig.xml，在 generatorConfig.xml 配置数据库连接、配置表等等。**注意：** 直接从课件目录下 practise/MyBatis 逆向工程目录中找到此文件拷贝过来修改即可。

2.4、运行插件

在 IDEA 的 Maven 窗口下中双击 mybatis-generator:generate 即可运行，运行注意观察控制台打印。



3、编写部门业务接口及实现

里面提供最基本 CRUD 方法。

```
package cn.wolfcode.service;

public interface IDepartmentService {
    void delete(Long id);
    void save(Department department);
    Department get(Long id);
    List<Department> listAll();
    void update(Department department);
}
```

```
package cn.wolfcode.service.impl;

public class DepartmentServiceImpl implements IDepartmentService {
    private DepartmentMapper departmentMapper;

    @Override
    public void delete(Long id) {
        departmentMapper.deleteByPrimaryKey(id);
    }

    @Override
```

```
    public void save(Department department) {
        departmentMapper.insert(department);
    }

    @Override
    public Department get(Long id) {
        return departmentMapper.selectByPrimaryKey(id);
    }

    @Override
    public List<Department> listAll() {
        return departmentMapper.selectAll();
    }

    @Override
    public void update(Department department) {
        departmentMapper.updateByPrimaryKey(department);
    }
}
```

三、Spring 集成 MyBatis（掌握）

1、配置数据库连接池

在 resources 目录下新建 applicationContext.xml，配置如下：

```
<!-- 关联 db.properties 文件 -->
<context:property-placeholder location="classpath:db.properties"/>

<!-- 配置 DataSource bean -->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-
method="init" destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>
```

2、配置 SqlSessionFactory bean

在 applicationContext.xml，配置如下：

```
<!-- 配置 SqlSessionFactory bean -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <!-- 关联主配置文件 目前可以不配置-->
    <property name="configLocation" value="classpath:mybatis-config.xml"/>
    <!-- 配置别名 若不用别名,可以不配置 -->
    <property name="typeAliasesPackage" value="cn.wolfcode.domain"/>
    <!-- 配置数据源-->
    <property name="dataSource" ref="dataSource"/>
    <!-- 关联 Mapper XML 可以不配置，前提编译 Mapper 接口字节码文件与 Mapper XML 文件在
一起，文件名也一样 -->
</bean>
```

3、配置 Mapper bean

在 applicationContext.xml，配置如下：

```
<!-- 配置 Mapper bean -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <!-- 指定 Mapper 接口所在包 -->
    <property name="basePackage" value="cn.wolfcode.mapper"/>
</bean>
```

4、配置 Service bean

修改 DepartmentServiceImpl，贴 IoC 和 DI 注解。

```
package cn.wolfcode.service.impl;

@Service
public class DepartmentServiceImpl implements IDepartmentService {

    @Autowired
    private DepartmentMapper departmentMapper;

    @Override
    public void delete(Long id) {
        departmentMapper.deleteByPrimaryKey(id);
    }

    @Override
    public void save(Department department) {
        departmentMapper.insert(department);
    }

    @Override
    public Department get(Long id) {
        return departmentMapper.selectByPrimaryKey(id);
    }

    @Override
    public List<Department> listAll() {
        return departmentMapper.selectAll();
    }

    @Override
    public void update(Department department) {
        departmentMapper.updateByPrimaryKey(department);
    }
}
```

在 applicationContext.xml，配置如下：

```
<!-- 配置 IoC DI 注解解析器 , 让 Spring 帮我们创建业务接口的实现类对象, 完成属性或者字段注入 -->
<context:component-scan base-package="cn.wolfcode.service.impl"/>
```

5、配置事务相关

在 applicationContext.xml，配置如下：

```
<!-- 配置事务管理器 WHAT-->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- 配置增强，包含 WHEN，并关联上面 WHAT-->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="get*" read-only="true"/>
        <tx:method name="select*" read-only="true"/>
        <tx:method name="query*" read-only="true"/>
        <tx:method name="count*" read-only="true"/>
        <tx:method name="list*" read-only="true"/>
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>

<!-- 配置 AOP -->
<aop:config>
    <!-- WHERE -->
    <aop:pointcut id="txPointcut" expression="execution(*
cn.wolfcode.service.impl.*ServiceImpl.*(..))"/>
    <!-- 关联 WHERE WHEN-->
    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut"/>
</aop:config>
```

6、编写单元测试类

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class DepartmentServiceTest {
    @Autowired
    private IDepartmentService departmentService;

    @Test
    public void testSave() {
        Department department = new Department();
        department.setName("公关部");
        department.setSn("DC");
        departmentService.save(department);
    }
}
```

四、集成 Spring MVC（掌握）

1、在 web.xml 配置端控制器和编码过滤器

```
<!-- 配置前端控制器 -->
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- 配置编码过滤器，针对 POST -->
<filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2、添加 Spring MVC 的配置文件

在 resources 目录新建 mvc.xml，配置如下：

```
<!-- IoC DI 注解解析器，配置扫描控制器。说人话：让 Spring 帮我们创建控制器对象，并注入 -->
<context:component-scan base-package="cn.wolfcode.web.controller"/>

<!-- 配置 MVC 注解解析器，时间注解，JSON 注解 -->
<mvc:annotation-driven/>

<!-- 处理静态资源 -->
<mvc:default-servlet-handler/>

<!-- 配置试图解析器 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

3、在 Spring MVC 的配置文件中引入 Spring 配置文件

在 mvc.xml，配置如下

```
<!-- 引入 applicationContext.xml -->
<import resource="classpath:applicationContext.xml"/>
```

4、编写 DepartmentController

```
package cn.wolfcode.web.controller;

@Controller
@RequestMapping("/department")
public class DepartmentController {
    @Autowired
    private IDepartmentService departmentService;

    // 处理查询所有部门请求
    @RequestMapping("/list")
    public String list(Model model){
        List<Department> departments = departmentService.listAll(qo);
        model.addAttribute("departments", departments);
        return "department/list"; // 找视图 /WEB-INF/views/department/list.jsp
    }

    // 处理删除请求
    @RequestMapping("/delete")
    public String delete(Long id){
        if (id != null) {
            departmentService.delete(id);
        }
        return "redirect:/department/list"; // 让浏览器重新发一次请求
    }

    // 去新增或去修改页面
    @RequestMapping("/input")
    public String input(Long id, Model model){
        if (id != null) { // 表示去修改
            Department department = departmentService.get(id);
            model.addAttribute("department", department);
        }
        return "department/input"; // 让浏览器重新发一次请求
    }

    // 新增或修改
    @RequestMapping("/saveOrUpdate")
    public String saveOrUpdate(Department department){
        if(department.getId()  == null){ // 新增
            departmentService.save(department);
        }else {
            departmentService.update(department); // 修改
        }
        return "redirect:/department/list"; // 让浏览器重新发一次请求
    }
}
```

5、拷贝部门 JSP 修改

到课件的 practise/views 目录拷贝 JSP 文件到项目的 webapp/views/department 目录下修改。

五、部门分页查询（掌握）

1、编写封装分页参数类

```
package cn.wolfcode.qo;

@Setter
@Getter
public class QueryObject {
    // 当前页
    private int currentPage = 1;
    // 每页条数
    private int pageSize = 5;
    public int getStart(){
        return (currentPage - 1) * pageSize;
    }
}
```

2、编写封装查询结果类

```
package cn.wolfcode.qo;

@Getter
public class PageResult<T> {

    private int currentPage; // 页面传的
    private int pageSize; // 页面传的
    private List<T> data; // 数据库查询
    private int totalCount; // 数据库查询

    private int totalPage; // 总页数
    private int prevPage; // 上一页
    private int nextPage; // 下一页

    public PageResult(int currentPage, int pageSize, int totalCount, List<T>
data) {
        this.currentPage = currentPage;
        this.pageSize = pageSize;
        this.data = data;
        this.totalCount = totalCount;

        // 加入判断，节省计算性能
        if(totalCount <= pageSize) {
            this.totalPage = 1;
            this.prevPage = 1;
            this.nextPage = 1;
            return;
        }

        this.totalPage = this.totalCount % this.pageSize == 0
            ? this.totalCount / this.pageSize : this.totalCount / this.pageSize
+ 1;
```

```
        this.nextPage = this.currentPage + 1 <= this.totalPage ?
this.currentPage + 1 : this.totalPage;
        this.prevPage = this.currentPage - 1 >= 1 ? this.currentPage - 1 : 1;
    }
}
```

3、给部门业务接口及实现添加分页功能

```
package cn.wolfcode.service;

public interface IDepartmentService {
    // ......
    PageResult<Department> query(QueryObject qo);
}
```

```
package cn.wolfcode.service.impl;

@Service
public class DepartmentServiceImpl implements IDepartmentService {

    @Autowired
    private DepartmentMapper departmentMapper;

    // ......

    @Override
    public PageResult<Department> query(QueryObject qo) {
        int count = departmentMapper.selectForCount(qo);
        if(count == 0){
            return  new PageResult<>(qo.getCurrentPage(), qo.getPageSize(),
                count, Collections.emptyList());
        }
        List<Department> departments = departmentMapper.selectForList(qo);
        return  new PageResult<>(qo.getCurrentPage(), qo.getPageSize(),
            count, departments);
    }
}
```

4、给部门 Mapper 接口及 Mapper XML 添加分页功能

```
package cn.wolfcode.mapper;

public interface DepartmentMapper {
    // ......

    int selectForCount(QueryObject qo);
    List<Department> selectForList(QueryObject qo);
}
```

```xml
<select id="selectForCount" resultType="java.lang.Integer">
    SELECT count(*)
    FROM department
</select>
<select id="selectForList" resultType="cn.wolfcode.domain.Department">
    SELECT id, name, sn
    FROM department
    LIMIT #{start}, #{pageSize}
</select>
```

5、修改部门控制器查询方法

```java
package cn.wolfcode.web.controller;

@Controller
@RequestMapping("/department")
public class DepartmentController {
    @Autowired
    private IDepartmentService departmentService;

    // 处理查询所有部门请求
    @RequestMapping("/list")
    public String list(Model model, QueryObject qo){
        PageResult<Department> pageResult = departmentService.query(qo);
        model.addAttribute("pageResult", pageResult);
        return "department/list"; // 找视图 /WEB-INF/views/department/list.jsp
    }

    // ......
}
```

6、修改部门 list.jsp

回显分页查询的部门数据。

六、员工增删改查（掌握）

1、使用逆向工程生成 Employee 实体类等

根据权限系统业务对象属性参照表文档建员工表，再使用逆向工程生成 Employee.java 、EmployeeMapper.java 、EmployeeMapper.xml。

```sql
CREATE TABLE `employee` (
    `id` bigint(20) NOT NULL AUTO_INCREMENT,
    `username` varchar(255) DEFAULT NULL,
    `name` varchar(255) DEFAULT NULL,
    `password` varchar(255) DEFAULT NULL,
    `email` varchar(255) DEFAULT NULL,
    `age` int(11) DEFAULT NULL,
    `admin` bit(1) DEFAULT NULL,
    `dept_id` bigint(20) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

生成的 Employee 类中 admin 字段注意改成 boolean 类型。

2、编写封装查询参数类

```java
package cn.wolfcode.qo;

@Setter
@Getter
public class EmployeeQueryObject extends QueryObject {
    private String keyword;
    private Long deptId; // 若这个值为 null 情况，就不根据部门过滤数据
}
```

3、拷贝部门业务类与控制器类修改

4、给员工 Mapper 接口及 Mapper XML 添加分页功能

```java
package cn.wolfcode.mapper;

public interface EmployeeMapper {
    // ......

    int selectForCount(QueryObject qo);
    List<Employee> selectForList(QueryObject qo);
}
```

```xml
<sql id="where_sql">
    <where>
        <if test="keyword != null">
            AND (name LIKE concat('%', #{keyword}, '%') OR email LIKE
concat('%', #{keyword}, '%'))
        </if>
        <if test="deptId != null">
            AND dept_id = #{deptId}
```

```
        </if>
    </where>
</sql>
<select id="selectForCount" resultType="java.lang.Integer">
    SELECT count(id)
    FROM employee
    <include refid="where_sql"/>
</select>
<select id="selectForList" resultMap="BaseResultMap">
    SELECT id, username, name, password, age, email, admin, dept_id
    FROM employee
    <include refid="where_sql"/>
    LIMIT #{start}, #{pageSize}
</select>
```

5、拷贝员工 JSP 修改

到课件的 practise/views 目录拷贝 JSP 文件到项目的 webapp/views/employee 目录下修改。

七、员工查询完善（掌握）

1、完善超管显示

超管显示为是或否，而不显示 true 或 false。修改员工 list.jsp，如下：

```
<td>${employee.admin ? '是' : '否'}</td>
```

2、完事部门显示

部门显示部门名称，而不是显示部门 id 值。

2.1、修改员工实体类

```
package cn.wolfcode.domain;

@Setter
@Getter
public class Employee {
    private Long id;
    private String username; // 登录用户名
    private String name; // 姓名
    private String password;
    private Integer age;
    private String email;
    private boolean admin;
    // 关联属性，不仅可以封装部门 id 可以封装部门名称
    private Department dept;
}
```

2.2、修改员工 Mapper XML

修改查询员工 SQL 及结果集处理的 resultMap。

```
 <resultMap id="BaseResultMap" type="cn.wolfcode.domain.Employee">
    <id column="id" property="id"/>
    <result column="username" property="username"/>
    <result column="name" property="name"/>
    <result column="password" property="password"/>
    <result column="age" property="age"/>
    <result column="email" property="email"/>
    <result column="admin" property="admin"/>
    <result column="d_id" property="dept.id"/>
    <result column="d_name" property="dept.name"/>
    <result column="d_sn" property="dept.sn"/>
</resultMap>

<sql id="where_sql">
    <where>
        <if test="keyword != null">
            AND (e.name LIKE concat('%', #{keyword}, '%') OR e.email LIKE
concat('%', #{keyword}, '%'))
        </if>
        <if test="deptId != null">
            AND e.dept_id = #{deptId}
        </if>
    </where>
</sql>

<select id="selectForCount" resultType="java.lang.Integer">
    SELECT count(e.id)
    FROM employee e
    <include refid="where_sql"/>
</select>
<select id="selectForList" resultMap="BaseResultMap">
    SELECT e.id, e.name, e.password, e.age, e.email, e.admin, e.dept_id, d.id AS
d_id, d.name AS d_name, d.sn AS d_sn
    FROM employee e
    LEFT JOIN department d ON e.dept_id = d.id
    <include refid="where_sql"/>
    LIMIT #{start}, #{pageSize}
</select>
```

2.3、修改员工 list.jsp

```
<td>${employee.dept.name}</td>
```

3、完善过滤查询

页面提供关键字输入栏及部门下来框，让用户输入关键字与选择部门，以便过滤查询员工数据，并回显查询条件。

3.1、修改员工控制器

```
package cn.wolfcode.web.controller;

@Controller
@RequestMapping("/employee")
```

```
public class EmployeeController {
    @Autowired
    private IEmployeeService employeeService;
    @Autowired
    private IDepartmentService departmentService;

    // 处理查询所有员工请求
    @RequestMapping("/list")
    // 把查询的方法的封装参数的形参改成 EmployeeQueryObject 类型，并贴上
@ModelAttribute("qo")
    public String list(Model model, @ModelAttribute("qo")EmployeeQueryObject qo)
{
        PageResult<Employee> pageResult = employeeService.query(qo);
        model.addAttribute("pageResult", pageResult);


        List<Department> departments = departmentService.listAll();
        // 查询所有部门往模型存入，提供数据，让员工列表页面显示出来以供用户选择
        model.addAttribute("departments", departments);

        return "employee/list";
    }

    // ......
}
```

3.2、修改员工 list.jsp

只要表单那块即可，提供搜索的关键字输入栏和部门选择下拉框，并使用 EL 表达式回显查询条件，如下：

```
<!-- 查询的表单 -->
<form action="/employee/list" method="post">
    <input type="hidden" name="currentPage" id="currentPage" value="1"/>
    关键字：<input type="text" name="keyword" value="${qo.keyword}"/>
    部门：<select name="deptId">
            <option value="">全部</option>
            <c:forEach items="${departments}" var="department">
                <option value="${department.id}" ${department.id == qo.deptId ?
'selected' : ''}></option>
            </c:forEach>
    </select>
    <input type="submit" value="查询"/>
</form>
```

八、员工新增修改完善（掌握）

1、员工新增完善

新增员工时，部门不要输入而是选择。

1.1、修改员工控制器

```
package cn.wolfcode.web.controller;
```

```
@Controller
@RequestMapping("/employee")
public class EmployeeController {
    @Autowired
    private IEmployeeService employeeService;
    @Autowired
    private IDepartmentService departmentService;

    @RequestMapping("/input")
    public String input(Long id, Model model){

        List<Department> departments = departmentService.listAll();
        // 查询所有部门往模型存入，提供数据，让员工新增修改时显示出来以供用户选择
        model.addAttribute("departments", departments);

        if (id != null) { // 表示去修改
            Employee employee = employeeService.get(id);
            model.addAttribute("employee", employee);
        }
        return "employee/input";
    }

    // ......
}
```

1.2、修改员工 input.jsp

把原来 input 改成 select，并使用 JSTL 遍历部门数据。

```
部门:<select name="dept.id">
    <c:forEach items="${departments}" var="department">
        <option value="${department.id}">${department.name}</option>
    </c:forEach>
</select>
```

1.3、修改员工 Mapper XML

因为之前我们把员工实体类的 Long 类型的 deptId 属性改成了 Department 类型的 department 属性，所以要修改员工保存的代码，不然报找不到属性的错误。

```
<insert id="insert" parameterType="cn.wolfcode.domain.Employee"
useGeneratedKeys="true" keyProperty="id">
    INSERT INTO employee (username, name, password, age, email, admin, dept_id)
    VALUES (#{username}, #{name}, #{password}, #{age}, #{email}, #{admin}, #
{dept.id})
</insert>
```

2、员工修改完善

2.1 修改的时候部门得回显

2.1.1、修改根据 id 查询的 SQL

```
<select id="selectByPrimaryKey" resultMap="BaseResultMap">
    SELECT e.id, e.username, e.name, e.password, e.age, e.email, e.admin,
e.dept_id, d.id AS d_id, d.name AS d_name, d.sn AS d_sn
    FROM employee e
    LEFT JOIN department d ON e.dept_id = d.id
    WHERE e.id = #{id}
</select>
```

2.1.2、修改员工 input.jsp

```
部门:<select name="dept.id">
    <c:forEach items="${departments}" var="department">
        <option value="${department.id}" ${department.id == employee.dept.id ?
'selected' : ''}></option>
    </c:forEach>
</select>
```

2.2、修改员工 Mapper XML

- 修改部门 id 值的获取。
- 去除密码的修改。

```
<update id="updateByPrimaryKey" parameterType="cn.wolfcode.domain.Employee">
    UPDATE employee
    SET username = #{username}
        name = #{name},
        age = #{age},
        email = #{email},
        admin = #{admin},
        dept_id = #{dept.id}
    WHERE id = #{id}
</update>
```

练习

```
CREATE TABLE `department` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `sn` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `employee` (
    `id` bigint(20) NOT NULL AUTO_INCREMENT,
    `username` varchar(255) DEFAULT NULL,
    `name` varchar(255) DEFAULT NULL,
    `password` varchar(255) DEFAULT NULL,
    `email` varchar(255) DEFAULT NULL,
    `age` int(11) DEFAULT NULL,
    `admin` bit(1) DEFAULT NULL,
    `dept_id` bigint(20) DEFAULT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- 练习一，根据以上表结构和数据，基于 Spring MVC + Spring + MyBatis，完成部门基本的 CRUD 和分页查询，完成员工基本的 CRUD、分页查询和过滤查询（根据名字和邮箱模糊查询，根据部门查询）。

```
CREATE TABLE `brand` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `product` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `price` decimal(10,2) DEFAULT NULL,
  `brand_id` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- 练习二，根据以上表结构和数据，基于 Spring MVC + Spring + MyBatis，完成品牌基本的 CRUD 和分页查询，完成产品基本的 CRUD、分页查询和过滤查询（根据名称模糊查询，根据价格范围查询，根据品牌查询）。