

国际物流云商系统第三天

一. 回顾

- 1.项目首页面执行过程分析
- 2.Spring Data JPA 及 DAO 实现
- 3.Page 分页组件
- 4.编写及配置 Service,Action
- 5 完成 DeptAction 类的分页操作

二. DeptAction 查看&更新&删除

1. 查看部门详情

```
/**
 * 查看详情
 */
@Action(value = "deptAction_toview", results = {
    @Result(name = "toview", location = "/WEB-INF/pages/sysadmin/dept/jDeptView.jsp") })
public String toview() throws Exception {
    // 1.调用业务方法，查询一个部门
    //Dept obj = deptService.get(model.getId());
    //Dept obj = deptService.findByDeptName("总裁办");
    Dept obj = deptService.findByDeptName("总裁办");

    // 2.将obj压入值栈中
    super.push(obj);

    return "toview";
}
```



2.进入部门新增页面

```
/**
 * 进入新增页面
 *
 * @return
 * @throws Exception
 */
@RequestMapping(value = "deptAction_tocreate", results = {
    @Result(name = "tocreate", location = "/WEB-INF/pages/sysadmin/dept/jDeptCreate.jsp") })
public String tocreate() throws Exception {
    // 1.加载状态为1的部门列表
    Specification<Dept> spec = new Specification<Dept>() {
        public Predicate toPredicate(Root<Dept> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };

    List<Dept> objList = deptService.find(spec);

    // 2.压入值栈
    super.put("deptList", objList);


    return "tocreate";
}
```

3.实现新增操作

```
/**
 * 实现新增
 */
@RequestMapping("deptAction_insert")
public String insert() throws Exception {
    // 调用业务方法，实现保存
    deptService.saveOrUpdate(model);
    return "alist";
}
```

4.进入更新页面

修改部门

上级部门：	船运部 
部门名称：	cgxaj

进入更新页面的实现代码如下：



```
/**
 * 进入修改页面
 *
 * @return
 * @throws Exception
 */
@RequestMapping(value = "deptAction_touupdate", results = {
    @Result(name = "touupdate", location = "/WEB-INF/pages/sysadmin/dept/jDeptUpdate.jsp") })
public String touupdate() throws Exception {
    // 1.加载状态为1的部门列表
    Specification<Dept> spec = new Specification<Dept>() {
        public Predicate toPredicate(Root<Dept> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    List<Dept> objList = deptService.find(spec);
    // 2.压入值栈
    super.put("deptList", objList);

    // 3.查询要修改的原有部门信息
    Dept obj = deptService.get(model.getId());

    // 4.将部门信息放入值栈中
    super.push(obj);

    return "touupdate";
}
```

5.实现修改操作

点击修改操作，执行的链接：

```
<a href="#" onclick="formSubmit('deptAction_update',this);this.blur();">保存</a></li>
```

在 DeptAction 类中添加一个 update 方法

```
/**
 * 实现修改操作
 */
@RequestMapping("deptAction_update")
public String update() throws Exception {
    // 1.先从数据库中查询原有部门信息
    Dept obj = deptService.get(model.getId());
    // 2.更新要修改的属性
    obj.setParent(model.getParent());
    obj.setDeptName(model.getDeptName());

    // 3.调用业务方法，修改部门对象
    deptService.saveOrUpdate(obj);

    return "alist";
}
```



6. 实现部门批量删除

查看

新增

修改

删除

```
<a href="#" onclick="formSubmit('deptAction_delete','_self');this.blur();">删除</a></li>
```

DeptAction 中 delete

```
/**
 * 删除,也支持批量删除
 */
@Action("deptAction_delete")
public String delete() throws Exception {
    // 1.调用业务方法,实现部门删除
    // deptService.deleteById(model.getId());

    // 批量删除
    deptService.delete(model.getId().split(", "));
    return "alist";
}
```

面试：具有相同名称的一组值，struts2 如何实现封装？

```
* model对象: Dept类型
* id: String
* 1.对于String类型, struts2会采用", "方式进行字符串连接 4028cdd255a3f7030155a3f835fd0000, 4028cdd955a079b60155a07cc6dc0000
* 2.对于Integer,Float,Date类型, struts2默认接收的是最后一个值
* 如: 客户端给id传了三个值, id=3 id=4 id=5----->服务器只能接收最后一个值,就是id=5
* 如何解决: ids=3 ids=4 ids=5 private List<Integer> ids ;//生成它的getter与setter
```

7.解决修改时的 BUG

如何解决修改时，自己为自己的父亲？

第一种方法：

```
public void saveOrUpdate(Dept model) {
    if(model.getParent()!=null && "".equals(model.getParent().getId())){
        model.setParent(null);
    }

    if(StringUtils.isEmpty(model.getId())){
        //说明是新增
        model.setState(1);
    }else{
        //更新
        //父部门就为自己
        if(model.getId().equals(model.getParent().getId())){
            model.setParent(null); //不允许自己为自己的父亲
        }
    }
    baseDao.saveOrUpdate(model);
}
```



第二种方法：

改造 jDeptUpdate.jsp 页面

```
<s:select name="dept.id" list="deptList"
    listKey="id" listValue="deptName"
    headerKey="" headerValue="--请选择--"
></s:select>
```

改为<select>标签

```
<c:forEach items="" var="dept" />
    <c:if test="${dept.id != id}"
        <option value="${dept.id}" >${dept.deptName}</option>
    </c:if>
</c:forEach>
```

第三种方法：

```
public String touupdate() throws Exception {
    //1.加载当前要更新的对象
    Dept dept = deptService.findDeptById(model.getId());
    super.push(dept);

    //2.下拉列表
    List<Dept> list = deptService.findAll();//查询所有的列表

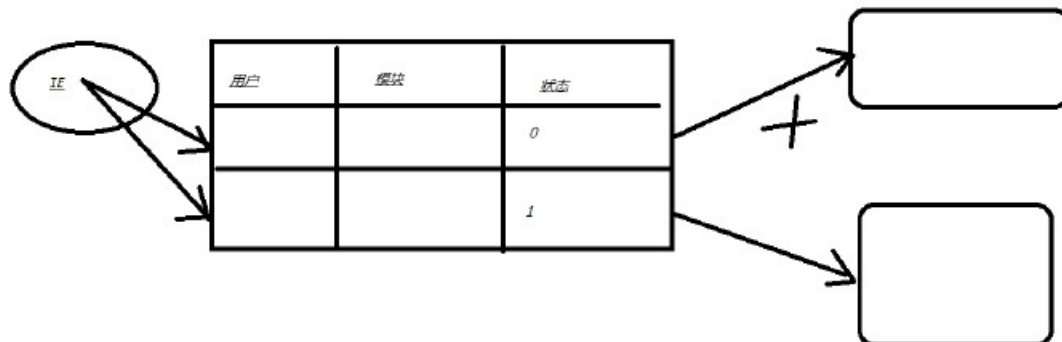
    //将当前放在值栈中要修改的对象，从部门列表中清除，目的是为了下拉列表中不会有当前放在值栈中的这个对象
    list.remove(dept);

    //将这个集合放入值栈的context中
    ActionContext.getContext().put("list", list);
    return "touupdate";
}
```

三. 传统认证与 RBAC 认证方式

1.传统认证方式

传统方式认证的过程就是查看状态字段，符合要求就可以访问功能模块

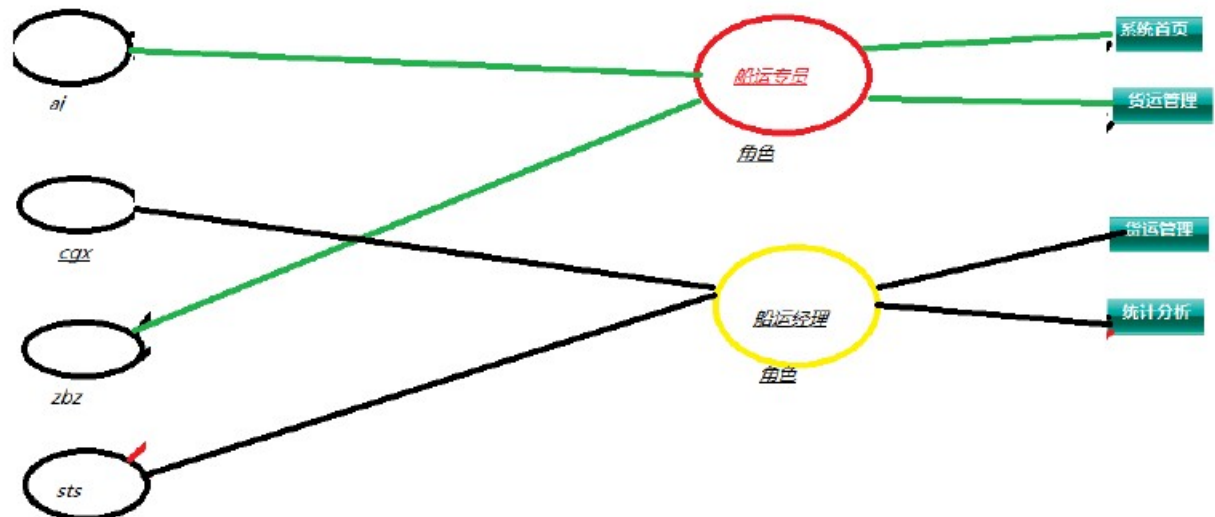


2.RBAC 认证方式

Role Base Access Controller :基于角色的访问控制

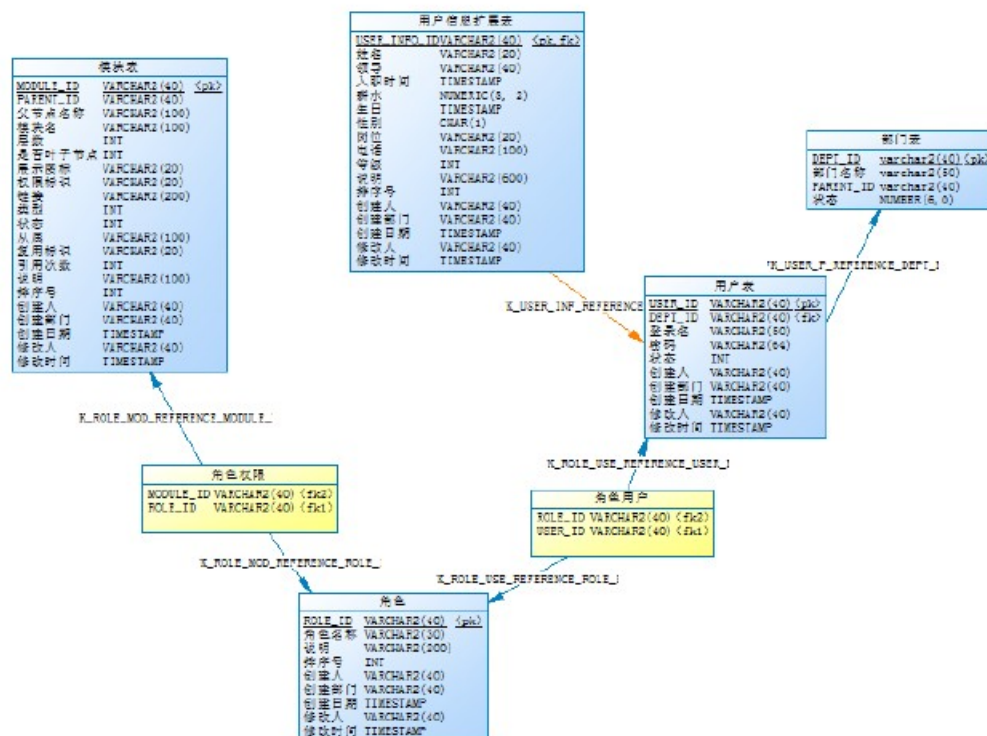
RBAC认证

现在主要用于各种系统的权限控制



四. RBAC 认证方式下的数据库设计

数据库设计:





将来登录时，要查上面的七张表

五. 实现用户与用户扩展信息的 CRUD

1. 用户实体类

- 抽取与用户扩展信息中的相同属性，放入父类中

@MappedSuperclass 代表抽取的父类注解，这样我们子类中继承父类时，这些注解也会继承过去。

```
@MappedSuperclass
public class BaseEntity implements Serializable {

    @Column(name="CREATE_BY")
    protected String createBy; //创建者的id
    @Column(name="CREATE_DEPT")
    protected String createDept; //创建者所在部门的id
    @Column(name="CREATE_TIME")
    @Temporal(TemporalType.TIMESTAMP)
    protected Date createTime; //创建时间
    @Column(name="UPDATE_BY")
    protected String updateBy; //修改者的用户id
    @Column(name="UPDATE_TIME")
    @Temporal(TemporalType.TIMESTAMP)
    protected Date updateTime; //更新时间
}
```

- User 直接继承 BaseEntity, 这样父类中的属性就可以复用代码了

```
@Table(name="USER_P")
@DynamicInsert(value=true)
@DynamicUpdate(value=true)
public class User extends BaseEntity {

    @Id
    @Column(name="USER_ID")
    @GeneratedValue(generator="system-assigned")
    @GenericGenerator(name="system-assigned", strategy="assigned")
    private String id; //编号

    @ManyToOne
    @JoinColumn(name="DEPT_ID")
    private Dept dept; //用户与部门 多对一

    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="USER_ID")
    private Userinfo userinfo; //一对一 用户与扩展信息

    @Column(name="USER_NAME")
    private String userName; //用户名
    @Column(name="PASSWORD")
    private String password; //密码
    @Column(name="STATE")
    private Integer state; //状态 0停用 1启用
}
```

- 用户扩展信息 Userinfo.java:



```
@Entity
@Table(name="USER_INFO_P")
@DynamicInsert(value=true)
@DynamicUpdate(value=true)
public class Userinfo extends BaseEntity {
    @Id
    @Column(name="USER_INFO_ID")
    @GeneratedValue(generator="system-assigned")
    @GenericGenerator(name="system-assigned",strategy="assigned")
    private String id;

    @Column(name="NAME")
    private String name;//真实姓名

    @ManyToOne
    @JoinColumn(name="MANAGER_ID")
    private User manager;//直属领导 用户与直属领导 多对一

    @Column(name="JOIN_DATE")
    @Temporal(TemporalType.TIMESTAMP)
    private Date joinDate;//入职时间

    @Column(name="SALARY")
    private Double salary;//薪水

    @Column(name="BIRTHDAY")
    @Temporal(TemporalType.TIMESTAMP)
    private Date birthday;//出生年月

    @Column(name="GENDER")
    private String gender;//性别

    @Column(name="EMAIL")
    private String email;//邮箱

    @Column(name="STATION")
    private String station;//岗位

    @Column(name="TELEPHONE")
    private String telephone;//电话号码

    @Column(name="DEGREE")
    private Integer degree;//等级

    @Column(name="REMARK")
    private String remark;//备注

    @Column(name="ORDER_NO")
    private Integer orderNo;//排序号
}
```

● 部门类 Dept 的修改

```
@Entity
@Table(name="DEPT_P")
public class Dept implements Serializable {
    @Id
    @Column(name = "DEPT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @OneToMany(mappedBy="dept")
    private Set<User> users = new HashSet<User>(0);//部门与用户 一对多

    @Column(name = "DEPT_NAME")
    private String deptName;

    @ManyToOne
    @JoinColumn(name="PARENT_ID")
    private Dept parent;//父部门 自关联 子部门与父部门 多对一关系

    @Column(name="STATE")
    private Integer state; //状态 1启用0停用 默认为1
}
```




2. 用户新增的业务逻辑

```
@Action(value = "userAction_tocreate", results = {
    @Result(name = "tocreate", location = "/WEB-INF/pages/sysadmin/user/jUserCreate.jsp") })
public String tocreate() throws Exception {
    // 1.加载状态为1的部门列表
    Specification<User> spec = new Specification<User>() {
        public Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    List<User> objList = userService.find(spec);
    // 2.压入值栈
    super.put("userList", objList);
    //3.查询所有可用部门
    Specification<Dept> spec2 = new Specification<Dept>() {
        public Predicate toPredicate(Root<Dept> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    //4.查询部门列表
    List<Dept> deptList = deptService.find(spec2);
    //5.将部门信息放入值栈中
    super.put("deptList", deptList);

    return "tocreate";
}

/**
 * 实现新增
 */
@Action("userAction_insert")
public String insert() throws Exception {
    // 调用业务方法，实现保存
    userService.saveOrUpdate(model);
    return "alist";
}
```

UserServiceImpl.java

```
@Override
public void saveOrUpdate(User entity) {
    if(UtilFuns.isEmpty(entity.getId())){
        entity.setState(1); //1代表启用，0代表停用
        //保证用户与用户扩展信息的id是相同的
        String id = UUID.randomUUID().toString();
        entity.setId(id);
        entity.getUserinfo().setId(id);
    }
    userDao.save(entity);
}
```

3.用户修改的 UserAction 实现

进入用户更新页面



```
@Action(value = "userAction_touupdate", results = {
    @Result(name = "touupdate", location = "/WEB-INF/pages/sysadmin/user/jUserUpdate.jsp") })
public String touupdate() throws Exception {
    // 1.查询要修改的原有部门信息
    User obj = userService.get(model.getId());
    // 2.将部门信息放入值栈中
    super.push(obj);
    // 3.加载状态为1的用户列表
    Specification<User> spec = new Specification<User>() {
        public Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    List<User> objList = userService.find(spec);
    //解决自己为自己的父部门
    objList.remove(obj);
    // 4.查询部门列表
    super.put("userList", objList);
    Specification<Dept> spec2 = new Specification<Dept>() {
        public Predicate toPredicate(Root<Dept> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    //5.压入值栈
    List<Dept> deptList = deptService.find(spec2);
    //将部门信息放入值栈中
    super.put("deptList", deptList);
    return "touupdate";
}
```

实现用户更新操作

```
/**
 * 实现修改操作
 */
@Action("userAction_update")
public String update() throws Exception {
    // 1.先从数据库中查询原有部门信息
    User obj = userService.get(model.getId());
    // 2.更新要修改的属性
    obj.setDept(model.getDept());
    obj.getUserinfo().setName(model.getUserinfo().getName());
    obj.setState(model.getState());

    // 3.调用业务方法，修改部门对象
    userService.saveOrUpdate(obj);
    return "alist";
}
```

4.用户删除

```
/**
 * 删除，也支持批量删除
 */
@Action("userAction_delete")
public String delete() throws Exception {
    // 批量删除
    userService.delete(model.getId().split(", "));
    return "alist";
}
```



六. 角色的 CRUD

当添加角色时，因为角色与用户之间存在的是多对多关系，所以我们可以使用 JPA 配置出这种关系，而且这种关系我们认为是双向的。因为将来可以通过用户得到他的角色信息，也可以通过角色去获取出这个角色分配给了哪些用户。

1. 添加 Role.java 的实体类

```
@Entity
@Table(name="ROLE_P")
@DynamicInsert(true)
@DynamicUpdate(true)
public class Role extends BaseEntity {
    @Id
    @Column(name="ROLE_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;//角色id

    @ManyToMany
    @JoinTable(name="ROLE_USER_P",joinColumns={@JoinColumn(name="ROLE_ID",referencedColumnName="ROLE_ID")},
    inverseJoinColumns={@JoinColumn(name="USER_ID",referencedColumnName="USER_ID")})
    private Set<User> users = new HashSet<User>(0);//角色与用户 多对多

    @Column(name="NAME")
    private String name;//角色名称

    @Column(name="REMARK")
    private String remark;//备注

    @Column(name="ORDER_NO")
    private Integer orderNo;//排序号
```

2. 修改 User 类与 Role 类配置多对多关系

在用户 User 类中添加 roles 集合并配置

```
@ManyToMany()
@JoinTable(name="ROLE_USER_P",joinColumns={@JoinColumn(name="USER_ID",referencedColumnName="USER_ID")},
    inverseJoinColumns={@JoinColumn(name="ROLE_ID",referencedColumnName="ROLE_ID")})
)
@OrderBy("ORDER_NO")
private Set<Role> roles = new HashSet<Role>();//用户与角色 多对多
```

在角色 Role 类中添加 users 集合并配置

```
@ManyToMany(mappedBy="roles")
private Set<User> users = new HashSet<User>(0);//角色与用户 多对多
```

3. 添加 RoleDao 接口

```
public interface RoleDao extends BaseDao<Role> {

}
```




4.进入 RoleService 编写

RoleService 接口

```
public interface RoleService extends BaseService<Role> {  
  
}
```

RoleServiceImpl 实现类

```
@Service  
public class RoleServiceImpl extends BaseServiceImpl<Role> implements RoleService {  
    private RoleDao roleDao;  
  
    @Autowired  
    public void init(RoleDao roleDao){  
        super.baseDao = roleDao;  
        this.roleDao = roleDao;  
    }  
  
}
```

5.RoleAction 类的编写

```
@Namespace("/sysadmin")  
public class RoleAction extends BaseAction implements ModelDriven<Role> {  
    private Role model = new Role();  
    public Role getModel() {  
        return model;  
    }  
    //分页组件的引入  
    private Page page = new Page();  
    public Page getPage() {  
        return page;  
    }  
    public void setPage(Page page) {  
        this.page = page;  
    }  
  
    //引入角色的业务逻辑  
    @Autowired  
    private RoleService roleService;
```




6.RoleAction 的分页查询

```
/**
 * 分页查询
 */
@RequestMapping(value = "roleAction_list", results = {
    @Result(name = "list", location = "/WEB-INF/pages/sysadmin/role/jRoleList.jsp") })
public String list() throws Exception {
    // 1.调用业务逻辑查询角色列表
    org.springframework.data.domain.Page<Role> page1 = roleService.findPage(null,
        new PageRequest(page.getPageNo() - 1, page.getPageSize()));

    // 2.设置page的相关属性
    super.parsePage(page, page1);
    // 3.设置page的url
    page.setUrl("roleAction_list");

    //4.将page放入值栈的栈顶
    super.push(page);

    return "list";
}
```

7.实现角色查看

```
/**
 * 查看
 */
@RequestMapping(value = "roleAction_toview", results = {
    @Result(name = "toview", location = "/WEB-INF/pages/sysadmin/role/jRoleView.jsp") })
public String toview() throws Exception {
    // 1.调用业务方法，实现查看详情
    Role role = roleService.get(model.getId());

    // 2.将role对象放入值栈中
    super.push(role);

    return "toview";
}
```

8.实现角色新增

```
/**
 * 进入新增页面
 */
@RequestMapping(value = "roleAction_tocreate", results = {
    @Result(name = "tocreate", location = "/WEB-INF/pages/sysadmin/role/jRoleCreate.jsp") })
public String tocreate() throws Exception {

    return "tocreate";
}
```



```
/**
 * 新增
 * @return
 * @throws Exception
 */
@Action(value="roleAction_insert")
public String insert() throws Exception {
    //1.调用业务方法，实现保存
    roleService.saveOrUpdate(model);
    return "alist";
}
```

修改 RoleServiceImpl 中的 saveOrUpdate()方法

```
@Override
public void saveOrUpdate(Role entity) {
    if(UtilFuns.isEmpty(entity.getId())){
        //新增
        //1.查询最大的orderNo,并且加1
        Integer maxOrderNo = (roleDao.queryMax()).intValue();
        entity.setOrderNo(maxOrderNo+1);
    }
    super.saveOrUpdate(entity);
}
```

添加 RoleDao 中的方法

```
//查询当前最大的orderNo取值
@Query(value="select max(orderNo) from Role")
public Long queryMax();
```

9.实现角色修改

```
/**
 * 进入修改界面
 */
@Action(value = "roleAction_toupdate", results = {
    @Result(name = "toupdate", location = "/WEB-INF/pages/sysadmin/role/jRoleUpdate.jsp") })
public String toupdate() throws Exception {
    // 1.调用业务方法，查询要修改的角色信息
    Role role = roleService.get(model.getId());
    // 2.保存到值栈中
    super.push(role);

    return "toupdate";
}

/**
 * 修改
 */
@Action(value="roleAction_update")
public String update() throws Exception {
    //1.加载要修改的原对象
    Role obj = roleService.get(model.getId());
    //2.设置修改的属性
    obj.setName(model.getName());
    obj.setRemark(model.getRemark());
    //3.调用业务方法，实现修改
    roleService.saveOrUpdate(obj);
    return "alist";
}
```



10. 实现角色删除

```
/**
 * 删除
 */
@Action(value="roleAction_delete")
public String delete() throws Exception {
    //1.调用业务方法，实现删除
    roleService.delete(model.getId().split(", "));
    return "alist";
}
```

七. 模块的 CRUD(作业)

1. 在角色 Role 类中添加 modules 集合并配置

```
@ManyToMany()
@JoinTable(name="ROLE_MODULE_P",
    joinColumns={@JoinColumn(name="ROLE_ID",referencedColumnName="ROLE_ID")},
    inverseJoinColumns={@JoinColumn(name="MODULE_ID",referencedColumnName="MODULE_ID")})
@OrderBy("ORDER_NO")
private Set<Module> modules = new HashSet<Module>(0); //角色与模块 多对多
```

2. 模块 Module 类及配置

```
@Entity
@Table(name="MODULE_P")
public class Module extends BaseEntity {
    @Id
    @Column(name="MODULE_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @ManyToMany(mappedBy="modules")
    private Set<Role> roles = new HashSet<Role>(0); //模块与角色 多对多

    @Column(name="PARENT_ID")
    private String parentId; //父模块的编号
    @Column(name="PARENT_NAME")
    private String parentName; //父模块的名称 冗余 用空间换时间
    @Column(name="NAME")
    private String name; //模块名
    @Column(name="LAYER_NUM")
    private Integer layerNum; //层数
    @Column(name="IS_LEAF")
    private Integer isLeaf; //叶子
    @Column(name="ICO")
    private String ico; //图片

    @Column(name="CPERMISSION")
    private String cpermission; //权限
    @Column(name="CURL")
    private String curl; //路径
    @Column(name="CTYPE")
    private Integer ctype; //菜单的类型: 主菜单, 左侧菜单
    @Column(name="STATE")
    private Integer state; //状态
    @Column(name="BELONG")
    private String belong; //从属于
    @Column(name="CWHICH")
    private String cwhich; //
    @Column(name="QUOTE_NUM")
    private Integer quoteNum; //引用次数
    @Column(name="REMARK")
    private String remark; //备注
    @Column(name="ORDER_NO")
    private Integer orderNo; //排序号
```



按照开发用户，角色同样的方式开发出模块管理

实现思路

首先开发模块的 **Domain**

其次开发模块管理的 **Dao** 接口，**Service** 接口及实现类，并进行配置

再次开发模块管理的 **Action** 类并配置

最后在 **jsp** 页面中测试 最终效果