

2012/7/7

数鹏通

开发框架参考文档



目录

1. 概述.....	3
2. 入门介绍.....	3
3. 核心技术.....	13
3.1 Action 类介绍	13
3.2 业务类介绍.....	19
3.3 DAO 层介绍	22
3.4 实体与枚举.....	24
3.5 模块关联.....	32
3.6 Excle 导出	35
3.7 工具类.....	37
3.8 JavaScript 介绍.....	40
3.9 国际化.....	43
3.10 配置文件.....	44
4. 其它.....	45
4.1 开发环境.....	45
4.2 编码规范.....	50
4.3 代码生成.....	51

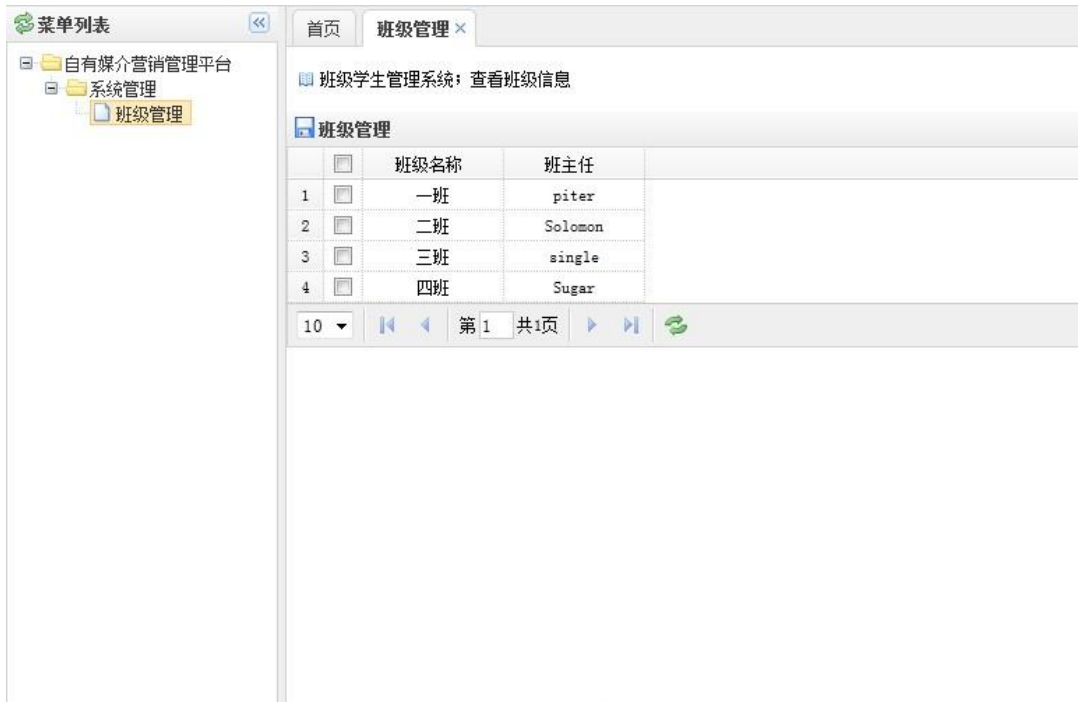
1. 概述

我们现在常常说用 JAVA 开发复杂麻烦 ,但具体复杂麻烦哪里 ,谁也没有说出个所以然。
最近十年 ,随着 MVC 框架 ,hibernate,spring 等框架的兴起和成熟 ,开发 J2EE 项目的效率相对来说比前有一定程度的提高。

这些框架提供 J2EE 的项目所需的基础功能 ,处理页面请求 ,数据库访问 ,容器管理 ,那我们还能通过什么手段提高开发效率吗 ?没错 ,就是从我们的业务入手 ,J2EE 项目最常见的业务模块的增删改查 ,我们的开发框架正是基于 SSH ,遵循 Don' t Repeat Yourself (DRY) 原则 ,通过对业务模块的增删改查的封装 ,把开发人员从这些重复又重复的劳动解放出来 ,从而把注意力集中在业务逻辑上。

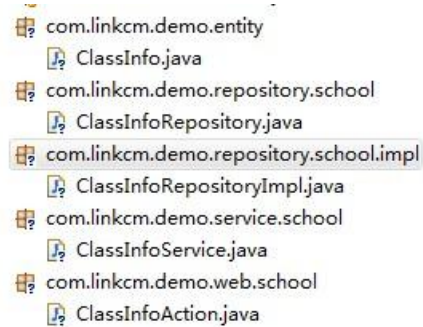
2. 入门介绍

现在 ,我们提供一个简单的教程 ,通过构建班级学生管理系统 (如图 2-1 所示) ,让开发人员逐步深入了解开发框架。



(图 2-1)

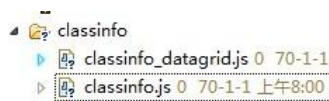
在我们的开发框架下构建班级学生管理系统（图 2-1）只需要 5 个包（图 2-2）和相应的 jsp（图 2-3）及 js（图 2-4）就足够了。



（图 2-2）



（图 2-3）



（图 2-4）

首先看下实体类 ClassInfo.java(图 2-5)。`@Entity` 表示这个为实体类。`@Table (name= "classinfo")` 中的 "classinfo" 就是数据库中的表明。`@Column (name= "classid")` 中的 classid 就是数据库中表的列名。实体就是这样与数据库相关联的。

```

@Entity
@Table(name = "classinfo")
public class ClassInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "classid")
    private Long id;
    @Column(name = "classname")
    private String classname;
    @Column(name = "teacher")
    private String teacher;

    public String getTeacher() {
        return teacher;
    }

    public void setTeacher(String teacher) {
        this.teacher = teacher;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getClassname() {
        return classname;
    }

    public void setClassname(String classname) {
        this.classname = classname;
    }
}

```

(图 2-5)

接下来是 ClassInfoRepository.java(图 2-6)和他的实现类 ClassInfoRepositoryImpl.java(图 2-7)。

这是简单的班级管理 ClassInfoRepository.java 只需要继承基类 BaseRepository<ClassInfo,Long>。

两个范型参数 ClassInfo , Long 分别代码实体 , ClassInfo 的主键类型。

```

import com.linkcm.core.dao.BaseRepository;
import com.linkcm.demo.entity.ClassInfo;

public interface ClassInfoRepository extends BaseRepository<ClassInfo, Long>, ClassInfoRepositoryCustom {
}

```

(图 2-6)

```

import com.linkcm.core.dao.BaseRepositoryImpl;

public class ClassInfoRepositoryImpl extends BaseRepositoryImpl<ClassInfo, Long> {
}

```

(图 2-7)

接着是逻辑处理类 ClassInfoService.java(图 2-8)。班级管理只需要继承 CoreService.java,

实现 getDao()方法，提供需要的 ClassInfoRepository，实现 getModule()用来记录日志。

```
@Component
@Transactional
public class ClassInfoService extends CoreService<ClassInfo, Long> {

    @Autowired
    private ClassInfoRepository classInfoRepository;

    @Override
    protected BaseRepository<ClassInfo, Long> getDao() {
        return classInfoRepository;
    }

    @Override
    public String getModule() {
        return "班级管理";
    }
}
```

(图 2-8)



提示：

ID 类型要对应。ClassInfoRepository 继承的 BaseRepository<ClassInfo, Long> ,ClassInfoRepositoryImpl 继承的 BaseRepositoryImpl<ClassInfo, Long>和 ClassInfoService 继承的 CoreService<ClassInfo, Long>他们的 Long 类型都是与实体里的 ID 类型相对应的。

下面是 Action 类。ClassInfoAction.java (图 2-9) 班级管理 Action 只需要继承 CoreAction.java

实现 getService()方法，提供需要的 ClassInfoService。

```
package com.linkcm.demo.web.school;

import org.apache.struts2.convention.annotation.Namespace;

@Namespace("/classinfo")
public class ClassInfoAction extends CoreAction<ClassInfo, Long> {

    private static final long serialVersionUID = 1L;

    @Autowired
    private ClassInfoService classInfoService;

    @Override
    public CoreService<ClassInfo, Long> getService() {
        return classInfoService;
    }
}
```

(图 2-9)

现在到直接与用户进行交互的 jsp，class-info.jsp（图 2-10）jsp 是负责分页展示数据的页面，事实上该页面只定义了 id 为“school_classinfo_datagrid”的 table 标签作为占位符，真正展示数据的是脚本是 classinfo.js 和 classinfo_datagrid.js。

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ include file="/common/taglibs.jsp" %>
<script type="text/javascript">var basePath='${ctx}';</script>
<script type="text/javascript" src="${ctx}/resources/js/app/classinfo/classinfo_datagrid.js"></script>
<script type="text/javascript" src="${ctx}/resources/js/app/classinfo/classinfo.js">
</script>
<div style="width:95%; height:auto; line-height:30px; margin:8px 4px 8px 4px;">
  <span style="width:20px; height:30px; background: url('${ctx}/resources/images/role_icon.png') no-repeat center center; float:left; font-weight:bold; font-size:12px; display:inline-block; margin-right:5px;">
    班级管理
  </span>
  <table id="school_classinfo_datagrid"></table>
</div>
```

(图 2-10)


最后就是 javascript 脚本（开发框架使用的页面技术是 jquery easyui）classinfo.js（图 2-12）和 classinfo_datagrid.js（图 2-11）。很多时候我们会遇到类似于显示部门员工，班级学生等时 js 出现大量的相同代码。基于这种状况，我们把可重用的代码生成了 classinfo_datagrid.js。在需要在班级管理界面显示学生时只需要调用 student_datagrid.js 里的方法。

```
$.ns('school.classinfo');

//数据列
school.classinfo.columns = [
  {title : "班级编码",width : '100',field : 'id',align : 'center',hidden : true},
  {title : "班级名称",width : '100',field : 'classname',align : 'center'},
  {title : "班主任",width : '100',field : 'teacher',align : 'center'}
];
```

(图 2-11)

首先定义页面要显示的数据列 school.classinfo.columns，这里有定义了三个字段，但班级编码作为数据库主键是不显示的。



提示：

IE 对代码要求比较严格。有一些 js 问题火狐等浏览器可能会编译通过，IE 就会出错。如（图 2-12）里的 dataColumn 如果给它最后一列加上逗号，那么火狐可以编译通过，但是 IE 就会出错。


```

$(document).ready(function() {
    var datagrid={
        url:basePath+'/classinfo/class-info!listAjax.action',
        title:"班级管理"
    };
    sy.datagrid("school_classinfo_datagrid",datagrid,school.classinfo.columns,true);
});

```

(图 2-12)

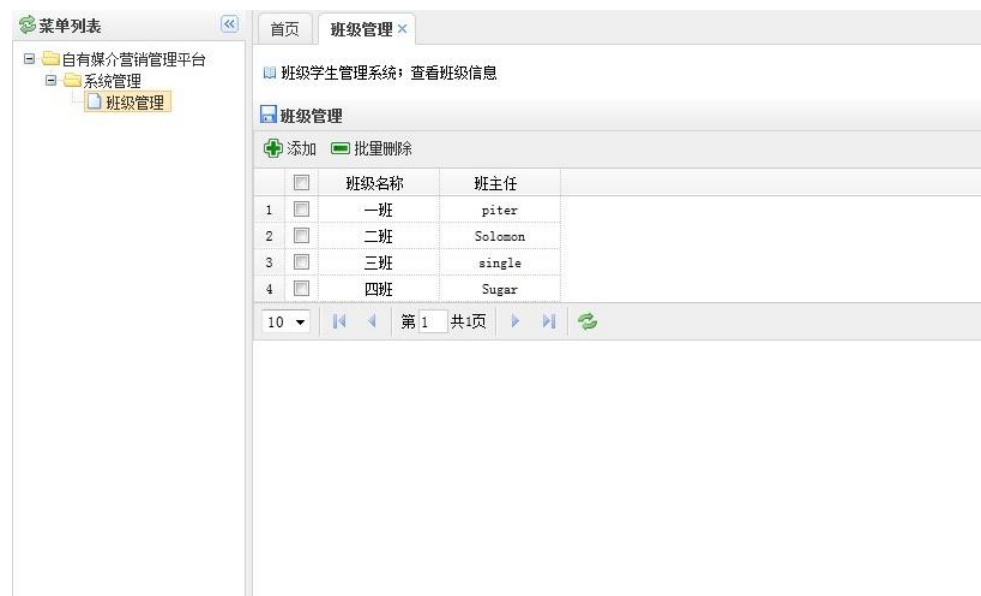
/classinfo/class-info!listAjax.action 是请求分页数据的 url , 返回为 json 对象 , sy.datagrid()的参数分别为数据表格在页面的占位符 ID , datagrid 对象 , 要显示的数据列 , true 参数代表每一行最左边增加一个复选框。。



提示：

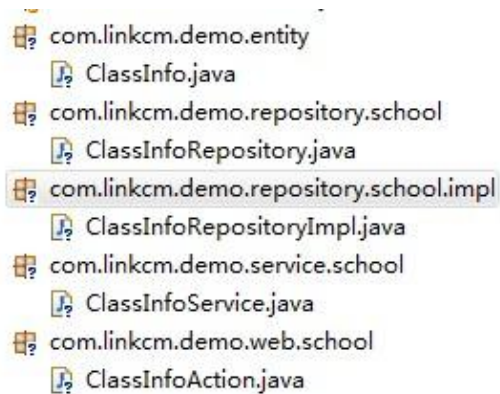
Strust2 对大小写是相当敏感的 , Action 以驼峰命名的一般用破折号连接原大写改为小写。如(图 2-9)Action 命名为 ClassInfoAction(图 2-11)的 Action 请求路径 , 是 classinfo/class-info!listAjax.action.

班级学生管理系统增添了添加及批量删除功能 (图 2-13)。

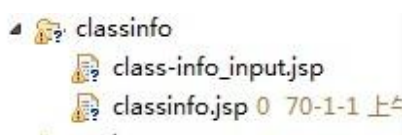


(图 2-13)

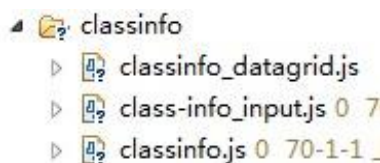
程序框架整体结构(图 2-14 到图 2-16)上多了 class-info_input.jsp 和 class-info_input.js. 看下他们的内部变化 (没有发生变化的就不再列出)。



(图 2-14)



(图 2-15)



(图 2-16)

在增加功能下增添多了 class-info_input.jsp (图 2-17)。同样需要导入相应的 js。

class-info_input.js。

```

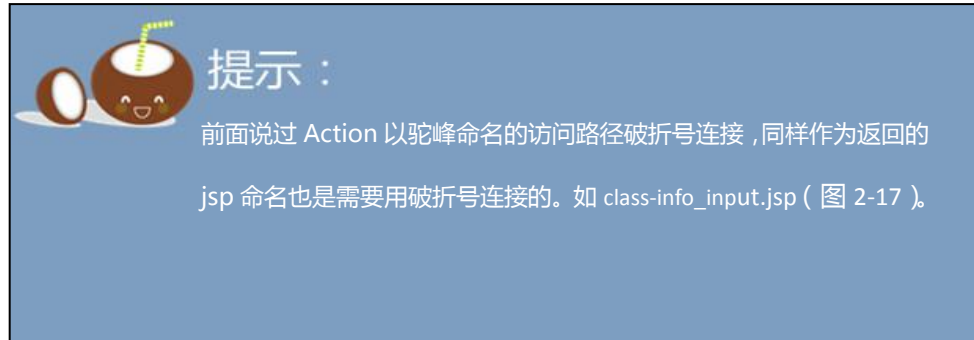
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ include file="/common/taglibs.jsp" %>

<script type="text/javascript">var basePath='${ctx}';</script>
<script type="text/javascript" src="${ctx}/resources/js/app/classinfo/class-info_input.js"></script>
<div class="easyui-layout" fit="true">
    <div region="center" align="center" style="padding-top:10px; color:#333;">
        <form id="school_classinfo_form" method="post" action="${ctx}/classinfo/class-info!saveByOperate.action">
            <input type="hidden" name="operate" value="${operate}"/>
            <input name="id" id="id" value="${id}" type="hidden"/>
            <table >
                <tr>
                    <td align="right"><s:text name="班级名称"></s:text></td>
                    <td align="left" width="70%">
                        <input name="classname" id="classname" value="${classname}" class="easyui-validatebox"/>
                    </td>
                </tr>
                <tr>
                    <td align="right"><s:text name="班主任"></s:text></td>
                    <td align="left" width="70%">
                        <input name="teacher" id="teacher" value="${teacher}" class="easyui-validatebox"/>
                    </td>
                </tr>
            </table>
            </form>
        </div>
        <div region="south" border="false" style="text-align: right;">
            <a id="school_classinfo_input" style="display:none; width: 65px; float: right;" class="easyui-linkbutton"
        </div>
    </div>

```

(图 2-17)

/classinfo/class-info!saveByOperate.action 为保存新增实体的路径 (operate 为 false 时保存修改后的实体)。



class-info.jsp (图 2-18) 添加一个 id 为 "school_classinfo_add" 的 div 层，作为弹出新增班级对话框的占位符。

```
<div id="school_classinfo_add" class="easyui-window" iconCls="icon-add" closed="true" collapsible="false" minimizable="false" |
```

(图 2-18)

同样的对应增添了 class-info_input.js(图 2-19)。

```
$(function() {  
    if(hasPermission("SAVE_CLASSINFO")){  
        $("#school_classinfo_input").show();  
    }  
    school.classinfo.add = function()  
    {  
        sy.trim('school_classinfo_form','classname','teacher');  
        sy.ajaxSubmit("school_classinfo_form","school_classinfo_add","school_classinfo_datagrid");  
    };  
});
```

(图 2-19)

sy.trim 方法用于去除表单值的前后空格，其所需参数 class-info_input.jsp 的 form id, 需要去空格的表单元素名。Sy.ajaxSubmit 用于提交表单，其参数是 class-info_input.jsp 的 form id, class-info.jsp 的 div 的 id(成功后关闭弹出窗口), 列表 JSP 的 datagrid 的 ID (重新刷新数据表格)。

classinfo_datagrid.js(图 2-20)增加了 school.classinfo.bulidButtons 方法，用于构建数据表格上的功能按钮。

```

school.classinfo.buildButtons = function() {
    var buttons = [];
    sy.buildAddButton(buttons, 'ADD_CLASSINFO', 'school_classinfo_add', {
        cache : false,
        title : "添加班级",
        href : basePath + '/classinfo/class-info!input.action',
        width : '280'
    });
    return buttons;
};

```

(图 2-20)

这里使用了 `sy.buildAddButton ()` , “ADD_CLASSINFO” 为权限代码 , 如果没有相应的权限则不会生成 button , “schhol_classinfo_add” 则为页面弹出窗口的占位符 , 最后一个参数是弹出窗口的属性 , 其中 `/classinfo/class-info!input.action` 返回的相应的添加页面 , `class-info_input.jsp`。

批量删除功能需要在 `classinfo_datagrid.js` 里加入 `school.classinfo.del` 方法 (图 2-21)。

```

//删除记录
school.classinfo.del = function(id)
{
    sy.postDel(id,basePath + "/classinfo/class-info!delete.action",'school_classinfo_datagrid');
};

```

(图 2-21)

`/classinfo/class-info!delete.action` 删除实体。 `Sy.postDel` 用于删除实体。其参数为：数据 ID , Action 路径 , 列表 JSP 的 `datagrid` 的 ID。

接着在 `classinfo_datagrid.js` 里的 `school.classinfo.bulidButtons` 加入下面代码 (图 2-22)。 “BATCH_DELETE_CLASSINFO” 为权限代码 , “sclool_classinfo_datagrid” 为数据表格 , 用来获取要删除的数据 , `school.classinfo.del` 为删除时所调用的函数。

```

sy.buildDelButton(buttons, 'BATCH_DELETE_CLASSINFO', 'school_classinfo_datagrid', school.classinfo.del);

```

(图 2-22)

下面是 `classinfo.js`(图 2-23):

```

var buttons = school.classinfo.buildButtons();
var datagrid={
    url:basePath+'/classinfo/class-info!listAjax.action',
    title:"班级管理",
    toolbar:buttons
};
sy.datagrid("school_classinfo_datagrid",datagrid,school.classinfo.columns,true);

```

(图 2-23)

首先调用 `school.classinfo.buildButtons()` 获取 `buttons`，然后在 `datagrid` 增加 `toolbar` 属性。

给班级学生管理系统添加操作（图 2-24）。



(图 2-24)

以下是增加修改功能下 class-info_input.jsp (图 2-25) 的代码变化。添加和修改用的是同一个页面, 在 class-info_input.jsp 加入 (图 2-25) 用 operate 进行判断, 当它为 true 时为添加, false 时为修改。

```
<form id="school_classinfo_form" method="post" action="${ctx}/classinfo/class-info!saveByOperate.action">
<input type="hidden" name="operate" value="${operate}"/>
<input name="id" id="id" value="${id}" type="hidden"/>
```

(图 2-25)

接下来是 classinfo.js 在增加修改功能后的改变,增加了 operate 操作 (图 2-26):

```
var operation = {title: "操作", width:'340',field:'opt',align:'center',  
formatter:function(value,rec,index){  
    var ops = [];  
    if(hasPermission("UPDATE_CLASSINFO")){  
        var updateOpt = '<a class="a-icon" title="'+ "记录修改" +' onclick="school.classinfo.postUpdate(\''+ rec.id + '\')"'  
ops.push(updateOpt);  
    }  
return ops.join('&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&');  
}
```

(图 2-26)

在 classinfo datagrid.js 里增加 school.classinfo.postUpdate (图 2-27) 方法

```
school.classinfo.postUpdate = function (id)
{
    $('#school_classinfo_add').window({cache:false,title: "修改", href:basePath+"/classinfo/class-info!input.action?operate=false&id="+id,width:'280'});
    $('#school_classinfo_add').window('open');
};
```

(图 2-27)

"/classinfo/class-info!input.action?operate=false&id "+id. 返回 class-info_input.jsp, operate 为 false (修改), 传 id。

增添删除功能唯一改变的就是 classinfo.js 在添加的 operate 后加入了添加了删除的 operate (图 2-28)。

[illegible]

这样，我们开发框架版的“Hello World”也就完成了。

想进一步了解 jquery easyui 可以访问：<http://www.jeasyui.com/>

3. 核心技术

在入门给大家介绍了框架最基本的结构及使用,在核心技术这一章将更深入介绍我们开发框架使用的各种技术及业务流程。

3.1 Action 类介绍

Action 类是用户请求和业务逻辑之间的桥梁,每个 Action 充当客户的一项业务代理。3.1 给大家讲述框架的 Action 流程,及其重要的方法。

1) 修改默认排序

我们的开发框架默认是按主键排序。

classid	classname	teacher
1	一班	piter
2	二班	Solomon
5	三班	single
4	四班	Sugar

(图 3.1-1)

+ 添加		- 批量删除	
	<input type="checkbox"/>	班级名称	班主任
1	<input type="checkbox"/>	一班	piter
2	<input type="checkbox"/>	二班	Solomon
3	<input type="checkbox"/>	四班	Sugar
4	<input type="checkbox"/>	三班	single

(图 3.1-2)

修改默认排序需要重写 getDefaultSort()方法。例如，classinfo 需要按 teacher 排序，需在 ClassInfoAction 里重写 getDefaultSort() (图 3.1-3)。

```
@Override
public Sort getDefaultSort() {
    return new Sort("teacher");
}
```

(图 3.1-3)

+ 添加		- 批量删除	
	<input type="checkbox"/>	班级名称	班主任
1	<input type="checkbox"/>	三班	Single
2	<input type="checkbox"/>	二班	Solomon
3	<input type="checkbox"/>	四班	Sugar
4	<input type="checkbox"/>	一班	piter

(图 3.1-4)

2) 增加查询功能

首先在 class-info.jsp 在入如下代码：

```

<div id="school_classinfo_search" align="center" style="display: none;padding-top:10px;color:#333;" class="easyui-window" iconCls="icon-search" closed="true" collapsible="false"
<form id="school_classinfo_search_form">
  <table>
    <tr>
      <td align="right"><input type="text" name="#<#>" /></td>
      <td align="right"><input type="text" name="filter_LIKE_classname" /></td>
    </tr>
    <tr>
      <td align="right"><input type="text" name="#<#>" /></td>
      <td align="right"><input type="text" name="filter_LIKE_teacher" /></td>
    </tr>
    <tr>
      <td colspan="2" align="right">
        <a class="easyui-linkbutton" iconCls="icon-ok" href="javascript:void(0)" onclick="school.classinfo.clear();" /><input type="text" name="global_clear" /></td>
        <a class="easyui-linkbutton" iconCls="icon-ok" href="javascript:void(0)" onclick="school.classinfo.search();" /><input type="text" name="global_query" /></td>
      </td>
    </tr>
  </table>
</form>
</div>

```

(图 3.1 -5)

里面定义了一个查询表单，有两个 input 元素，分别用于根据班级名和班主任来过滤数据，name 属性 `filter_LIKE_classname` 通过下划线分为三部分，filter 标识这是一个过滤条件，LIKE 是操作符，其它操作符的还包括 EQ，NE，LE，LT，GE，GT，分别代表等于，不等于，少于等于，少于，大于等于，大于。最后的 classname 是要过滤字段。

然后在 `classinfo_datagrid.js` 加入：

```

//清空查询条件
school.classinfo.clear = function(){
  $('#school_classinfo_search_form')[0].reset();
  $('#school_classinfo_datagrid').datagrid("load",{});
};
//查询
school.classinfo.search =function(){
  sy.search('school_classinfo_datagrid','school_classinfo_search_form');
};

```

(图 3.1 -6)

`sy.search ()` 的参数分别为数据表格（获取过滤后的数据重新刷新），查询表单（获取这个单下的查询条件）。

最后还要构造“查询”按钮，在 `classinfo_datagrid.js` 的 `school.classinfo.buildButtons` 函数里加入：

```

sy.buildQueryButton(buttons,'QUERY_CLASSINFO','school_classinfo_search',{cache: false,title: "查询按钮", width:"250"});

```

(图 3.1 -7)

运行结果如图：



(图 3.1 -8)

3) 重写 listAjax

我们在前面用 esayui 构造数据表格的时候就提到请求数据的路径是 /classinfo/class-info!listAjax.action。其实这个 action 通过基类 CoreAction 的 listAjax() 方法实现 ,该方法返回的是分页数据。现在系统需要记录每一个数据请求 ,默认的 listAjax() 是没有这个功能 ,我们可以重写 listAjax() (图 3.1-9) 来记录数据请求并返回分页数据。

```
public String listAjax() throws Exception {
    try {
        System.out.println("记录管理——查询数据");
        EasyUiDataGrid entity = getService().searchForAjax(getPageable(), ServletUtils.getParameters());
        WebUtils.renderJson(entity);
    } catch (Exception e) {
        failure(e);
    }

    return null;
}
```

(图 3.1-9)

代码最后一句 return null ,因为这是一个 ajax 请求 ,返回的结果是 json 数据而不是页面。其余的代码都是都被一个 try..catch 包围 ,这样做的好处就能捕获业务层抛出的异常并通过 failure()向请求方返回错误信息 ,这样页面就能显示一个相对友好的提示 ,实际编程中我们也必须遵循这个规范。再来看 getService().searchForAjax(getPageable(),

ServeltUtils.*getParameters()* , *getService()*返回的是 ClassInfoService , 从父类继承过来的 *searchForAjax* 返回分页数据 EasyUiDataGrid , *getPageable()*方法获取请求的每页的数据量 , 页数并构造有排序的 Pageable 对象 , ServeltUtils.*getParameters()*获取查询过滤条件。

4) CoreAction 的属性与方法

一般来说我们的自定义 action 都是继承基类 CoreAction , 我们要善于利用 CoreAction 的已有的属性和方法 , 减少工作量和代码量。CoreAction 的属性如下 :

```
private T entity;  
private String sort;  
private String order;  
private PK id;  
private String ids;  
  
// 每页显示的页数  
private int rows = 10;  
  
// 当前页  
private int page = 1;  
  
// 操作标志,默认是新增  
private boolean operate = true;
```

(图 3.1-10)

entity 是实体 , 新增修改时都会把页面 post 过来的值都设到 entity 里面。sort , order 分别排序字段和排序方式 , 如果 datagrid 开启了这个功能 , 请求数据时就会把这两个字段传到 action 来 (排序枚举字段时 , statusDesc 的这样字段在实体是不存在的 , 要增加一个 orderField 属性 , 如果字段名和实体一致的则不用增加 orderField , 例如:{title: sys_user_statusDesc,width:'100',field:'statusDesc',orderField:'status',align:'center',sortable:true})。id 是主键 , 当修改数据时 , 页面会先传要数据的 id 过来 , action 再根据 id 查出数据并赋值到 entity 属性 , 页面再显示要修改的数据。当删除数据时 , 页面会把要删除的 id 传到 ids , 如果是批量删除 , 字符串以逗号分隔。rows 和 page 就是分页数据的定

义。opeate 是操作标志，默认是新增。以上的属性都有对应的 setter，getter 方法，实际编程中可以灵活运用。现来介绍基类的一些方法，如下表：

名称	功能
success(String msg)	对页面的 ajax 请求返回一条成功信息
failure(String msg)	对页面的 ajax 请求返回一条失败信息
failure(Exception e)	对页面的 ajax 请求返回 e.getMessage()，推荐使用
list()	返回列表页
listAjax()	返回分页数据
findAll()	返回所有数据，以数组形式返回
findAllDataGrid()	返回所有数据
input()	返回新增或者修改页面
saveByOperate()	新增或者修改数据
getEntityInstance()	使用反射来生成一个实体
prepareModel()	这个方法很重要，如果新增数据时 action 的 id 属性为 null，修改时（请求修改页面而不是提交修改数据）id 是页面传过来的值，prepareModel 根据 id 来生成一个新的或者从数据库里查询出一个实体并赋值到 entity 属性
getIdArray()	把属性 ids 转换成 id 数组
delete()	删除数据
getModel()	返回 entity 属性

5) 模型驱动开发

CoreAction 实现了 ModelDriven，Preparable 两个接口，通过它们我们可以实现模型驱动开发，例如修改班级功能，在 class-info_input.jsp 里面，我们使用\${classname}而不是 \${entity.classname}就可以访问班级名称。下面介绍其中的详细过程，首先 ModelDriven 只有一个 getModel()方法，每个请求都会调用这个方法，如果返回结果不是 null 值，就尽量把参数值设到返回结果也就是实体里，或者把实体的属性值直接存到 ValueStack。但是 getModel()里只是简单返回 entity 属性，也就是 null 值，这是不可能实现以上功能，这时就需要用到 Preparable 接口，这个接口特别的地方是：假设我们的请求是调用 action 的 xxx 方法，而 action 里又有一个 prepareXxx()，那在处理请求前就首先调用 preareXxx()，所以 input()，saveByOperate()都有对应的 prepareInput()，prepareSaveByOperate()，里面再调用 prepareModel()，entity 属性就这样被赋值了，也就可以实现我们的模型开发。



3.2 业务类介绍

在前面的章节我们已经介绍过 Action 的基本用法和 CoreAction 处理请求的流程，下面我们继续探究处理业务的 Service。

1) 业务校验

业务校验是每个业务模块必不可少的部份，现在我们通过增加对班级名称的唯一性校验来说明业务类的校验过程。首先，当新增或者保存一个实体的时候，会调用 CoreAction 的 saveByOperate()，接着就调用 CoreService 的 save(entity,operate)方法，里面除了保存实体这个基本的操作，还有一个回调方法 beforeSave(entity,operate)，我们自定义的业务 service，这里是 ClassInfoService，就可以通过重写 beforeSave 来实现业务，如下图：

```
protected void beforeSave(ClassInfo entity, boolean operate) {  
    if (isEntityUniqueById(entity.getId(), "className", entity.getClassname())) {  
        throw new ServiceException("班级名称已存在");  
    }  
}
```

(图 3.2 -1)

isEntityUniqueById () 的第一个参数是实体的 ID，后面的是一个可变参数，形式为属性名，属性值.....，可以有多个这样的名值对。这个方法会根据实体的 ID 生成两种不同的查询语句，如果 ID 为 null (新增)，产生的语句如下：

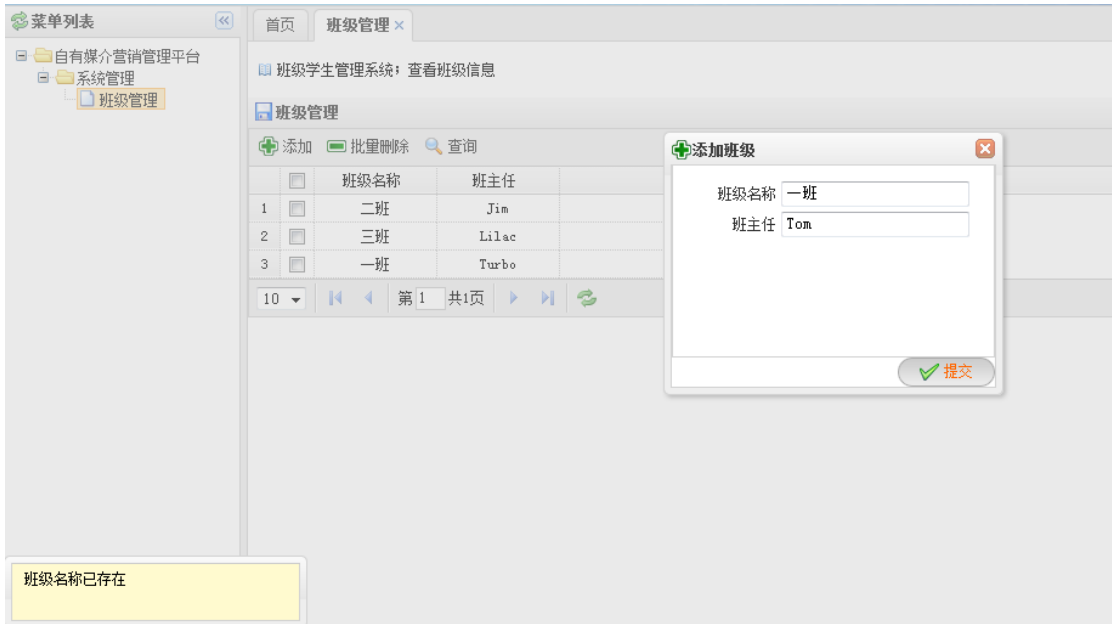
```
select classinfo0_.classid as classid9_, classinfo0_.classname as classname9_,  
classinfo0_.teacher as teacher9_ from classinfo classinfo0_ where className=?
```

如果 ID 不为 null (修改)，产生以下语句：

```
select classinfo0_.classid as classid9_, classinfo0_.classname as classname9_,
classinfo0_.teacher as teacher9_ from classinfo classinfo0_ where className=?
and classinfo0_.classid<>?
```

如果查询到结果则返回 true，证明当前的字段不是唯一，从而要抛出一个业务异常，save()

方法再往外抛，最后被 action 层捕获，把异常信息返回客户端，结果如下：



(图 3.2 - 2)

这里不得不说就是异常处理机制 在自定义业务方法或者重写 CoreService 的业务方法时候，所有业务代码都必须被 try..catch 包围：

```
try {
    beforeDelete(this.getEntity(id));
    getDao().delete(id);
} catch (Exception e) {
    throw handleException("删除失败", e);
}
```

(图 3.2 - 3)

在处理异常的时常使用了 handleException()方法，里面会分别处理 ServiceException 和 Exception，捕获到 ServiceException，例如上面的班级名称检验，简单地把异常返回；捕获到其它的异常，例如空指针，IO 错误，数据库异常，首先就要用 Logger 来记录日志，然后再转换成 ServiceException 返回。这样做的好处是业务异常或者底层代码错误都能给用户一个最友好的提示而不是一个令人生畏的异常栈。另外值得注意就是，就算业务方法只

是简单地调用 dao 的查询方法，哪怕是简单的一句，也最好用上面的 try...catch 包围，因为随着系统的运行，数据会越来越多，原来不出错的查询到最后也会错误的可能性，有了 try...catch，就一而终地保证了提示的友好性。

2) 公共方法

CoreService 也和 CoreAction 一样，提供了一些易用的公共方法，如下表：

名称	功能
beforeSave(T entity)	保存实体的回调方法
beforeSave(T entity, boolean operate)	保存实体的回调方法，operate 为 true 时是新增，false 时是修改
beforeDelete(T t)	删除实体的回调方法
isEntityUnique(Object... paramsArray)	校验属性值是否唯一，与 isEntityUniqueById(null, paramsArray)等效
isEntityUniqueById(PK idValue, Object... paramsArray)	根据主键校验属性值是否唯一
getEntity(PK id)	根据主键获取实体
exists(PK idValue)	判断实体已经存在
isEndow(Class entityClz, String property, Object value)	判断实体有没有被其它实体关联，entityClz 为其它实体，property 为当前实体与其它实体的关联字段名称，value 是当前实体的关联字段值
getLogger()	返回日志类
getModule()	返回模块名
save(T entity)	保存实体
save(T entity, boolean operate)	保存实体
save(T[] entitys)	保存实体数组
save(T[] entitys, boolean operate)	保存实体数组
save(Collection<T> entitys)	保存实体集合
save(Collection<T> entitys, boolean operate)	保存实体集合
delete(PK id)	根据主键来删除实体
delete(T entity)	删除实体
delete(T[] entitys)	删除实体数组
delete(Collection<T> entitys)	删除实体集合
searchForAjax(final Pageable pageable, final List<QueryParam> filters)	返回 EasyUi 分页数据
search(final Pageable pageable, final List<QueryParam> filters)	返回 Page 形式分页数据
findAll()	获取所有实体
findAll(Sort sort)	获取指定排序的所有实体
handleException(String	处理异常

message,Exception e)	
----------------------	--

无论保存实体或者批量保存实体（数组，集合），都会保证调用对应的 `beforeSave()`，就是说，在批量保存中会多次调用 `beforeSave()`，只要其中一个 `beforeSave()` 抛出异常，事务都能得到回滚。批量删除实体也是一样的道理。

3.3 DAO 层介绍

目前为止，数据库持久层使用的两大主流技术无非就是 ORM 框架和原生的 jdbc，我们的开发框架对它们都有着不错的支持，下面逐一进行介绍。

1) Spring-Data-Jpa

这里我们使用的 ORM 框架是 Spring-Data-Jpa（底层实现是 hibernate），最大的优点就是可以根据方法名字来解析成 HQL 查询语句，有效提高持久层访问开发工作的效率。例如，现在学生班级管理系统中需要按班级名字查询班级，可以在 `ClassInfoRepository` 接口中加入以下代码：

```
public ClassInfo findByClassname(String classname);
```

（图 3.3 -1）

或者根据多个字段来查询：

```
public List<ClassInfo> findByClassnameAndTeacher(String classname, String teacher);
```

（图 3.3 -2）

这里返回类型 `List<ClassInfo>`，Spring-Data-Jpa 贴心地帮我们自动匹配返回结果，甚至在注释型查询中可以根据 HQL 返回 `String`，`Long` 等类型。当查询业务比较复杂时，例如表关联，直接用方法名来查询就行不通了，这时可以使用注释查询，如下图：

```
@Query("select c.classname from ClassInfo c where c.classname=? and c.teacher=? ")  
public List<String> findClassInfo(String classname, String teacher);
```

（图 3.3 -3）

前面在 CoreService 提到的 isEntityUniqueById() , isEndow() 其实都是调用了 BaseRepositoryImpl 对应的方法，这些方法就连注释查询也实现不了，这时通过扩展接口就实现更复杂的查询，首先增加一个扩展接口 ClassInfoRepositoryCustom：

```
public interface ClassInfoRepositoryCustom {  
    public List<Map<String, Object>> findAllClassName();  
}
```

(图 3.3 - 4)

而且扩展接口必须遵循命名规范，在已有的 Reposiotory 名字+ “Custom”。另外 ClassInfoRepositoryImpl 还要实现 ClassInfoRepositoryCustom，代码如下：

```
@SuppressWarnings("unchecked")  
@Override  
public List<Map<String, Object>> findAllClassName() {  
    return em.createQuery("select new Map(c.className as className) from ClassInfo").getResultList();  
}
```

(图 3.3 - 5)

对 Spring-Data-Jpa 我们暂时介绍到这里，想进一步了解可以访问 <http://static.springsource.org/spring-data/data-jpa/docs/current/reference/html/>

2) 原生 JDBC

有时为了多表连接，或者为了提高性能时不可避免会使用到原生的 JDBC，我们的开发框架有两种方法可以使用 JDBC。

a) 和扩展接口类似，首先在扩展接口加入自定义方法接口，然后在实现类里面注入

BaseJdbcDao，如下图：

```
@Autowired  
private BaseJdbcDao baseJdbcDao;  
  
@Override  
public List<Map<String, Object>> findAllClassName() {  
    return baseJdbcDao.findBySql("select classname as classname from classinfo");  
}
```

(图 3.3 - 6)

b) 另外一种就是新建一个实现类，这个实现类直接继承 BaseJdbcDao，然后在业务类注入这个新的实现类。

这里提倡第一种做法，这样程序员只和一个接口打交道，也隐藏了底层的实现。还有 JDBC 查询传统的做法会把查询结果转成 VO，这样就需要增加一个 VO 类和一个 Transformer 类，这里就没必要这样做，因为返回 Map<String,Object>无论 JSP 或者 JSON 都可以很好地支持。

最后介绍 BaseJdbcDao 的几个比较重要的方法，如下表：

名称	功能
findBySql(String sql,final Object... values)	通过 sql 查询，返回 List
findBySql(String sql, List<QueryParam> filters, final Object... values)	通过 sql 查询，返回 List
findPageBySql (Pageable pageable,String sql, final Object... values)	通过 sql 查询，返回 Page
findPageBySql (Pageable pageable,String sql, List<QueryParam> filters, final Object... values)	通过 sql 查询，返回 Page
getJdbcTemplate()	获取 JdbcTemplate
getDataSource()	获取 DataSource

3.4 实体与枚举

ORM 框架通过实体来实现 JAVA 对象与数据库表及字段的映射，而枚举又担当了实体类常量的角色，本章通过增加学生管理功能来介绍实体与枚举。

1) 实体校验

首先增加学生实体类，如下图：

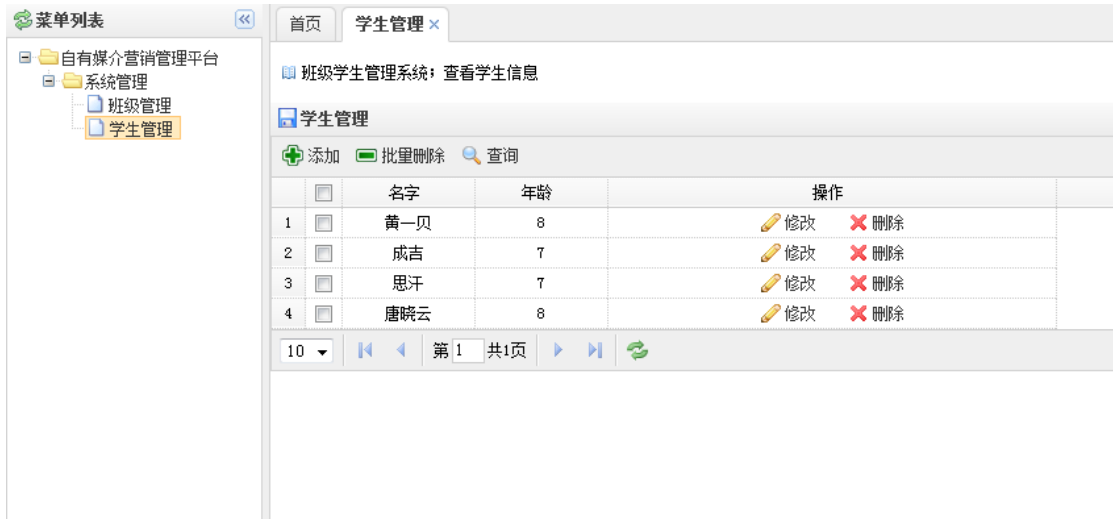

```
@Entity
@Table(name = "student")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "student_id")
    private Long id;
    @Column(name = "name")
    private String name;
    @Column(name = "age")
    private Integer age;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

(图 3.4 - 1)

接着与班级管理功能类似，增加 action 类，service 类，dao 类，jsp，js 等，运行界面如下：



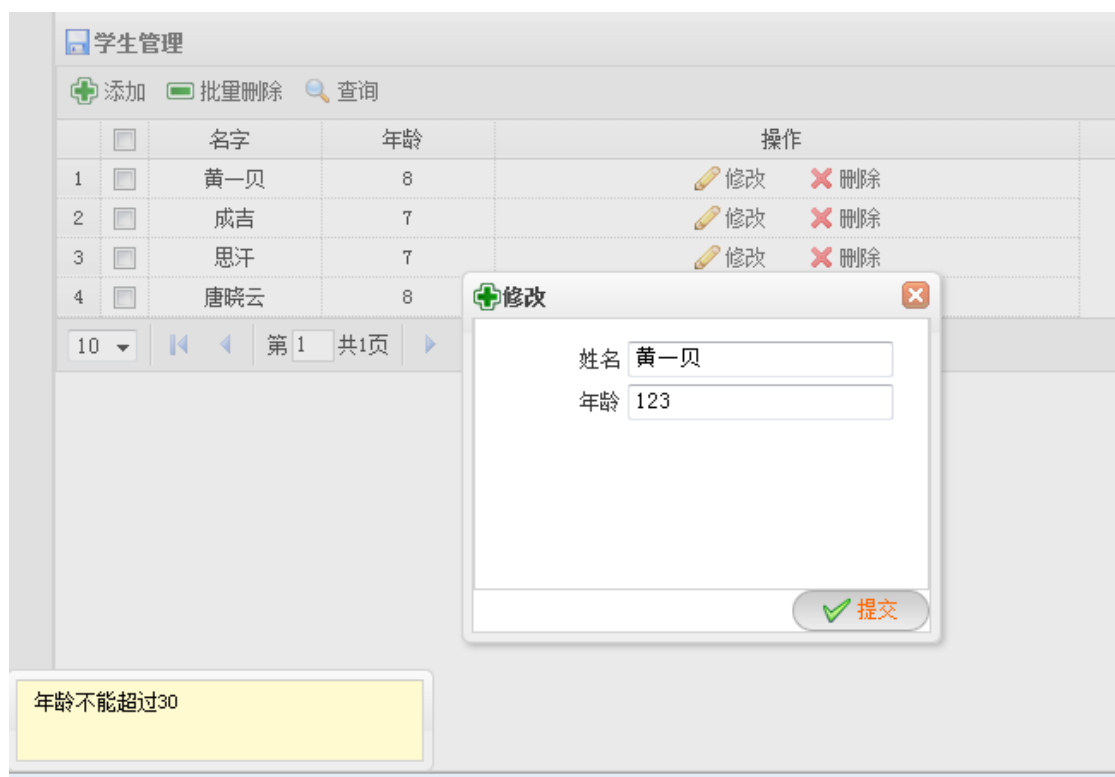
(图 3.4 - 2)

为了提高用户体验性，前台的 JS 校验是必不可少的，但是在用户大有可能直接在功能地址加上参数的形式直接提交来绕过前台的校验，还有通过 excel 批量导入数据也要对实体进行校验，这样一来，后台对实体的校验也成为系统是否安全的一个标准。现在，我们对姓名，年龄两个字段进行校验，如下图：

```
@Validation(name = "姓名", nullable = false, min = 2, max = 50)
@Column(name = "name")
private String name;
@Validation(name = "年龄", nullable = false, min = 6, max = 30)
@Column(name = "age")
private Integer age;
```

(图 3.4 - 3)

注解 Validation 标志了被注解的字段需要校验，该注解包括 6 个属性，name 为名称，nullable 为是否为空，min 为最小值（当属性为 String 类型为最小长度），max 为最大值（当属性为 String 类型为最大长度），email 为是否邮件，regex 为正则表达式，例如“[A-Z]+::名称必须为大写字母”，以“::”分为表达式，校验不通过时提示信息两部分。工具类 CoreUtils 的 validateEntity() 可以对实体进行校验，校验不通过则抛出 ServiceException。运行结果如图：



(图 3.4 - 4)

一般来说，validateEntity() 不需要显式调用，它在 action 类的 saveByOperate() 里面并在调用业务类的 save() 之前调用了，如果重写了 saveByOperate()，或者是 excel 批量导入这种特殊业务就需要显式调用 validateEntity()。

2) VoEntity

当 action 类使用业务类的 searchForAjax () 方法时，该方法返回了一个 EasyUiDataGrid 的分页数据，默认是实体对象，然后 action 类以 json 形式把 EasyUiDataGrid 返回浏览器

时会调用实体每个以 get 开头的方法，也就是说，有 getter 方法的实体属性必定会返回到页面，这样一来，假设一个实体具有 20 多个属性，甚至超过，而页面的列表页只需要用到个位数的属性，或者实体中有大对象或者集合之类的属性而列表页又不需要展示这些属性，这种情形，action 类和业务类的这种默认行为无疑是会增加了服务器的性能和网络负担。

实体类通过实现 VoEntity 并返回列表页感兴趣的数据，如下图：

```
@Override
public Object returnVo() {
    Map<String, Object> vo = new HashMap<String, Object>();
    vo.put("name", this.getName());
    vo.put("age", this.getAge());
    return vo;
}
```

(图 3.4 - 5)

returnVo()返回的 Object，这里是一个 HashMap (也可以是其它可以被转换成 JSON 的对象)，就会在 EasyUiDataGrid 里代替实体。

3) 枚举

编程过程中无可避免会使用常量，例如性别 0，1，展示时为男，女，在没有引入枚举前，这种数值与自然语言之间的转义会遍布整个系统：页面，业务代码，报表。下面通过在学生实体增加性别来介绍枚举的使用。首先，在 com.linkcm.demo.entity.type 下新增性别枚举 SexType，如下图：

```

public enum SexType {
    male(0, "男"), female(1, "女");

    public final int value;

    private final String desc;

    public static final Transformer transformer = new AbstractTransformer() {
        public String getDesc(Object type) {
            return SexType.getDesc(CoreUtils.object2int(type));
        }
    };

    private SexType(int value, String desc) {
        this.value = value;
        this.desc = desc;
    }

    public String getDesc() {
        return I18nUtils.getMessage(desc);
    }

    public static String getDesc(int type) {
        for (SexType enumType : SexType.values()) {
            if (enumType.value == type) {
                return enumType.getDesc();
            }
        }
        return "" + type;
    }
}

```

(图 3.4 - 6)

SexType 的构造方法可以 value , desc 两个参数 , value 为原始值 , desc 为要转义的自然语言。transformer 可以对 hql 或者 sql 查询出来的 Map 数据进行转义 , 这样查询语句或者就不用对常量的转义进行硬编码 , 提高了代码的可读性。接着在学生实体类加入性别属性 , 还在 student_input.jsp 加入以下代码 :

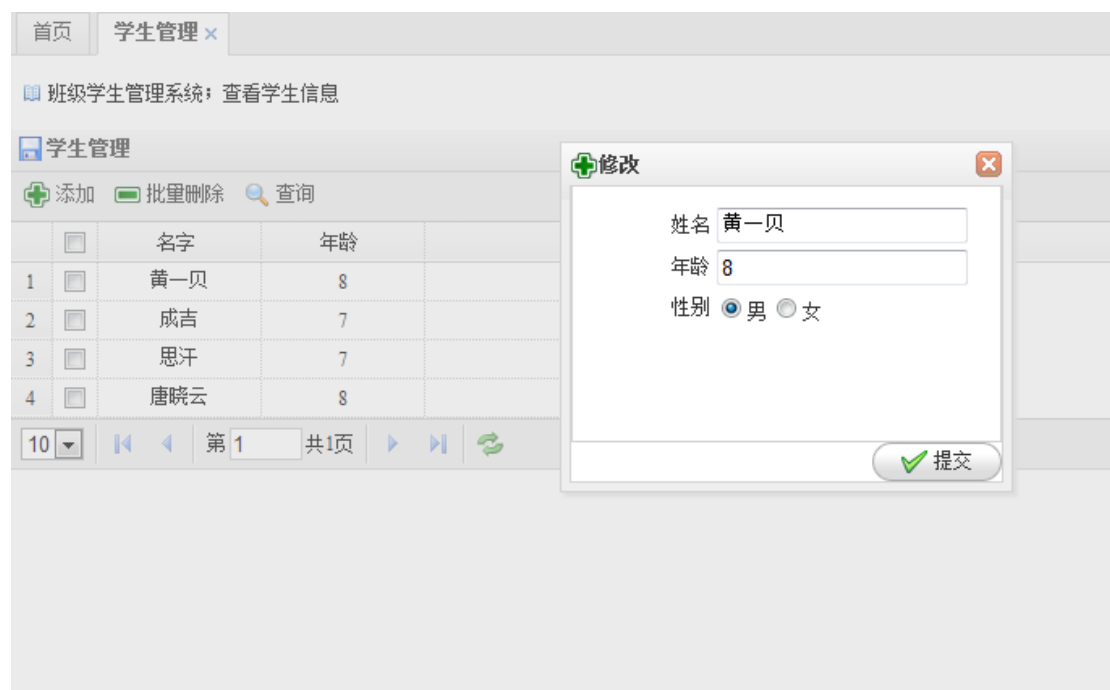
```

<tr>
    <td align="right" width="30%"><s:text name="性别"></s:text></td>
    <td align="left" width="70%">
        <s:radio name="sex" list="@com.Linkcm.demo.entity.type.SexType@values()" listKey="value" listValue="desc"
        </td>
</tr>

```

(图 3.4 - 7)

运行结果如图 :



(图 3.4 - 8)

然后在为学生列表页加上性别字段，上面说过，实体转换成 JSON 时以 get 开头的方法都会被调用，我们在学生实体加入以下代码：

```
public String getSexDesc() {
    return SexType.getDesc(sex);
}
```

(图 3.4 - 9)

在 student_datagrid.js 加入以下代码：

```
var dataColumns = [
    {title : "名字",width : '100',field : 'name',align : 'center'},
    {title : "年龄",width : '100',field : 'age',align : 'center'},
    {title : "性别",width : '100',field : 'sexDesc',align : 'center'}
];
```

(图 3.4 - 1 0)

运行结果如图：



(图 3.4 - 1 1)

下面通过重写业务类的 searchForAjax()来介绍枚举类的 transformer 的使用，首先新建一个 dao 扩展类及实现类，如图：

```
public interface StudentRepositoryCustom {  
    public Page<Map<String, Object>> findStudent(Pageable pageable, List<QueryParam> filters);  
}
```

(图 3.4 - 1 2)

```
public class StudentRepositoryImpl extends BaseRepositoryImpl<Student, Long> implements StudentRepositoryCustom {  
    @Autowired  
    private BaseJdbcDao baseJdbcDao;  
  
    @Override  
    public Page<Map<String, Object>> findStudent(Pageable pageable, List<QueryParam> filters) {  
        StringBuilder sb = new StringBuilder(  
            "select student_id as id,name as name,age as age,sex as sexDesc from student where 1=1");  
        for (QueryParam filter : filters) {  
            if (filter.getName().equalsIgnoreCase("name")) {  
                sb.append(" and name=").append(filter.getValue()).append("");  
            } else if (filter.getName().equalsIgnoreCase("age")) {  
                sb.append(" and age=").append(filter.getValue()).append("");  
            }  
        }  
        return baseJdbcDao.findPageBySql(pageable, sb.toString());  
    }  
}
```

(图 3.4 - 1 3)

或者使用 BaseJdbcDao 的 findPageBySql(Pageable pageable, String sql, List<QueryParam> filters)，这样不用对 filters 遍历来进行 sql 的拼接。还有对查询条件进行直接拼接的话，恶意入侵者就有机会利用 SQL 注入获取重要数据甚至入侵系统。代码如下：

```

public class StudentRepositoryImpl extends BaseRepositoryImpl<Student, Long> implements StudentRepositoryCustom {

    @Autowired
    private BaseJdbcDao baseJdbcDao;

    @Override
    public Page<Map<String, Object>> findStudent(Pageable pageable, List<QueryParam> filters) {
        String sql = "select student_id as id,name as name,age as age,sex as sexDesc from student";
        return baseJdbcDao.findPageBySql(pageable, sql, filters);
    }

}

```

(图 3.4 - 14)

因为查询参数都是以 String 类型存放在 QueryParam，如果过滤的字段有日期类型的，要在查询语句里把对应的字段先转换成字符串类型，数值型的字段则不用转换。

这样一来，业务类就可以使用 findStudent() 来获取学生列表，重写业务类的 searchForAjax()，如图：

```

@SuppressWarnings({ "unchecked", "rawtypes" })
public EasyUiDataGrid searchForAjax(final Pageable pageable, final List<QueryParam> filters) {
    try {
        Page pageEntity = studentRepository.findStudent(pageable, filters);
        SexType.transformer.transfer("sexDesc", pageEntity);
        return EasyUiUtils.getEasyUiDataGrid(pageEntity);
    } catch (Exception e) {
        throw handleException("获取列表失败", e);
    }
}

```

(图 3.4 - 15)

通过 SexType.transformer 就把结果中的 sexDesc 字段从 0，1 转换成男，女，这样查询语句和页面都不再参与常量的转义。

这里再说个题外话，在开发过程中我们常常碰到这样的场景，有两个列表页展示的是同一个表的数据，但是数据的类型有区别，例如已审批列表和未审批列表，下面通过学生列表页只展示性别为男的学生作为示例，代码如下：

```

public class StudentRepositoryImpl extends BaseRepositoryImpl<Student, Long> implements StudentRepositoryCustom {

    @Autowired
    private BaseJdbcDao baseJdbcDao;

    @Override
    public Page<Map<String, Object>> findStudent(Pageable pageable, List<QueryParam> filters) {
        String sql = "select student_id as id,name as name,age as age,sex as sexDesc from student where sex=?";
        return baseJdbcDao.findPageBySql(pageable, sql, filters, SexType.male.value);
    }

}

```

(图 3.4 - 16)

和之前的实现的区别是在 sql 语句在加入" sex=?" 并把条件值 SexType.male.value 传到 findPageBySql 方法里面，运行结果如图：

学生管理						
<div><div>添加</div><div>批量删除</div><div>查询</div></div>						
	<input type="checkbox"/>	名字	年龄	性别	班级	操作
1	<input type="checkbox"/>	aaa	12	男		修改 删除
2	<input type="checkbox"/>	ccc	11	男		修改 删除
3	<input type="checkbox"/>	qqq	8	男		修改 删除
4	<input type="checkbox"/>	xxx	12	男		修改 删除
5	<input type="checkbox"/>	rrr	11	男		修改 删除
6	<input type="checkbox"/>	fff	21	男		修改 删除
7	<input type="checkbox"/>	aaa	11	男		修改 删除
8	<input type="checkbox"/>	aaa	11	男		修改 删除
9	<input type="checkbox"/>	aaa	11	男		修改 删除
<div><div>10</div><div> </div><div>第 1 共 1 页</div><div></div></div>						

(图 3.4 - 17)

3.5 模块关联

我们这里把模块定义为对一个实体的增删改查的操作功能集合(当然还有其它一些功能)，例如班级管理是一个模块，学生也是一个模块，模块与模块之间常常是互相联系的，例如学生应该属于某个班级，班级又可以查看它的所有学生，让我们继续修改代码来完成这些功能。首先在学生实体在加上 classId 字段，因为学生列表页要显示班级，我们在实体相应增加 getClassTitle()方法，代码如下：

```
public Long getClassId() {
    return classId;
}

public void setClassId(Long classId) {
    this.classId = classId;
}

public String getClassTitle() {
    return CoreUtils.code2value(classId, ClassInfo.class, "id", "classname");
}
```

(图 3.5-1)

在 getClassTitle() 通过 code2value() 把 classId(code) 转义成 classname(value)。接着修改 student_datagrid.js 以显示 classTitle 字段：


```
var dataColumns = [
    {title : "名字",width : '100',field : 'name',align : 'center'},
    {title : "年龄",width : '100',field : 'age',align : 'center'},
    {title : "性别",width : '100',field : 'sexDesc',align : 'center'},
    {title : "班级",width : '100',field : 'classTitle',align : 'center'}
];
```

(图 3.5-2)

学生管理除了要显示班级字段,还要在增加或修改页面构造一个班级下拉框,确定学生是属于哪个班级,这里使用 struts2 的 select 标签,如图:

```
<tr>
    <td align="right"><s:text name="班级"></s:text></td>
    <td align="left">
        <s:select name="classId" list="classMap" theme="simple"></s:select>
    </td>
</tr>
```

(图 3.5-3)

这时, StudentAction 要提供 classMap 给 select 标签,如图:

```
public Map<Object, Object> getClassMap() {
    return JspUtils.getChooseMap(ClassInfo.class, "id", "classname");
}
```

(图 3.5-4)

通过 JspUtils.getChooseMap()把实体 ClassInfo 的 id, classname 以名值对形式返回,构造出来的下拉框有 headerValue “请选择”, JspUtils 的 getTableMap ()就没有 headerValue,运行结果如图:

(图 3.5-5)

最后，在班级列表上增加一个查看班级学生功能。首先在 classinfo.js 增加“班级学生”操作：

```
if(hasPermission("QUERY_STUDENT")){
    var pwdOpt = '<a class="a-icon" title="班级学生" onclick="school.classinfo.student(\'' + rec.id + '\')"'
    ops.push(pwdOpt);
}
```

(图 3.5-6)

再增加对应的单击事件：

```
school.classinfo.student=function (classId){
    var datagrid={
        height:280,
        url:basePath+'/student/student!listAjax.action?filter_EQS_classId='+classId,
        pageNumber : 1
    };
    sy.datagrid("school_classinfo_student_datagrid",datagrid,school.student.columns);
    $("#school_classinfo_student_data").window({cache:false,title: '班级学生', height:315});
    $("#school_classinfo_student_data").window("open");
};
```

(图 3.5-7)

datagrid 的 pageNumber 设为 1，确保查看不同的班级学生时都从第一页开始。然后在 class-info.jsp 引入 student_datagrid.js 脚本并增加弹出窗口的占位符：

```
<div id="school_classinfo_student_data" class="easyui-window" iconCls="icon-add" closed="true" collapsible="false"
</div>
```

(图 3.5-8)

运行结果如图：



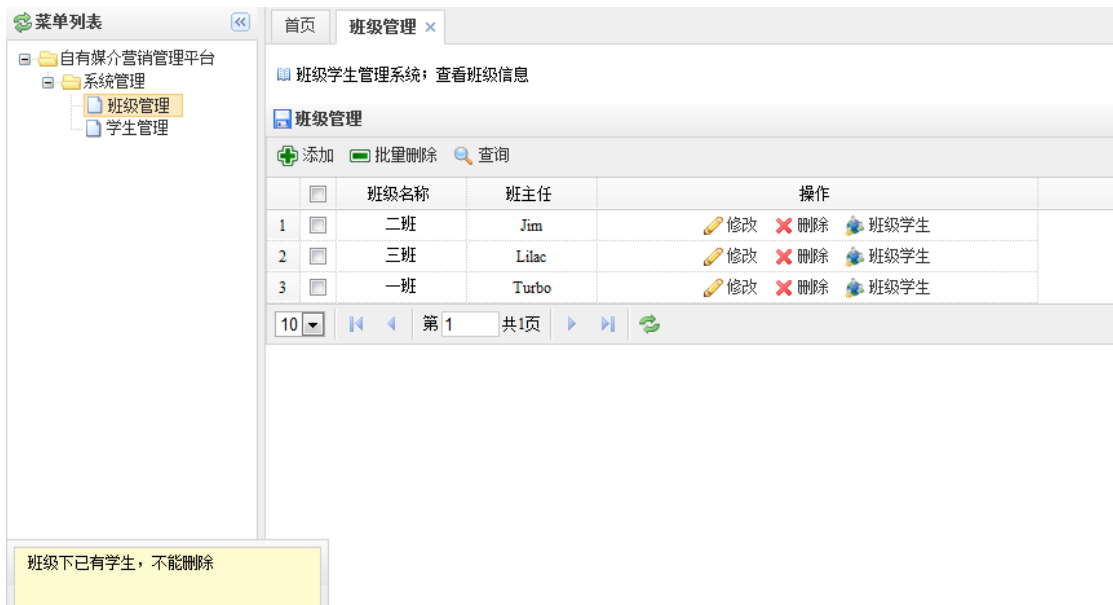
(图 3.5-9)

还有对删除班级增加判断，如果班级下已经有学生就不能够被删除，通过重写回调方法 beforeDelete()来实现以上需求，代码如下：

```
protected void beforeDelete(ClassInfo entity) {
    if (isEndow(Student.class, "classId", entity.getId())) {
        throw new ServiceException("班级下已有学生，不能删除");
    }
}
```

(图 3.5-10)

在 beforeDelete()方法里面使用了 isEndow()来判断班级下是否有学生 ,如果有则抛出业务异常。结果如图：



(图 3.5-11)

3.6 Excle 导出

开发框架使用了 jxl.jar 作为 excel 导出的工具类库，它最大的特点是不用操作单元格，而是直接基于模版导出，具有容易实现和输出样式丰富的优点。下面我们通过为学生增加导出功能来介绍 jxl 的使用。首先，在 student_datagrid.js 增加导出按钮，如图：

```
sy.buildExportButton(buttons, 'EXPORT_STUDENT', 'school_student_search_form', basePath+"/student/student!export.action");
```

(图 3.6-1)

“ school_sudent_search_form ” 为查询表单 (前面的查询也是使用这个表单) ，

"/student/student!export.action"为导出功能的路径。然后在 StudentAction 里面增加实例变量

List<Student> students 和 export()，如图：

```
private List<Student> students;

public List<Student> getStudents() {
    return students;
}

public String export() {
    try {
        students = getService().search(new PageRequest(0, Integer.MAX_VALUE, getSort()),
            ServletUtils.getParameters()).getContent();
    } catch (Exception e) {
        LOG.error("导出学生失败", e);
    }

    return "export";
}
```

(图 3.6-2)

这样 Struts2 暂时是识别不 export()是一个导出 excel 的方法，需要在 Action 头部加入以下注解：

```
@Results({ @Result(name = "export", type = "excel", location = "/resources/excel/students.xls") })
```

(图 3.6-3)

然后在/webapp/resources/excel 目录下增加 excel 模版 students.xls，如图：

	A	B	C	D	E
1	天河中学学生名册				
2	姓名	年龄	性别	班级	
3	<jx:forEach items="\${students}" var="student">				
4	\${student.name}	\${student.age}	\${student.sexDesc}	\${student.classTitle}	
5	</jx:forEach>				
6					
7					
8					
9					
10					
11					
12					

(图 3.6-4)

在 excel 模版里面对 ValueStack 的变量使用和 JSP 相关不多(所以变量 students 在 action 里面要有 getter 方法)，这里使用了<jx:forEach>标签对 students 进行遍历。导出结果如图：

	A	B	C	D
1	天河中学学生名册			
2	姓名	年龄	性别	班级
3	黄一贝	8	男	一班
4	成吉	7	男	一班
5	思汗	7	女	一班
6	唐晓云	8	女	二班
7				
8				

(图 3.6-5)

想进一步了解 jxl 可以访问 <http://jxls.sourceforge.net/>

3.7 工具类

开发框架提供了几大类的工具来提高效率，下面逐一进行介绍。

✓ SecurityUtils

提供了获取当前登陆用户资料的方法。

名称	功能
getId()	返回当前登陆用户 ID
getName()	返回当前登陆用户名

✓ ServletUtils

ServletUtils 的作用是获取页面请求的查询参数。

名称	功能
getParameters()	获取查询参数
getParametersPlusUserId(String filed, OperateType operate)	获取查询参数，额外增加根据用户 ID 过滤的查询参数
getParametersPlusUsername(String filed, OperateType operate)	获取查询参数，额外增加根据用户名过滤的查询参数

✓ JspUtils

用于构造有 headerValue 的下拉框数据，struts2 标签支持 headerValue 但不能把 headerValue 国际化。

名称	功能
getTableMap(Class<?> clz, String key, String value)	根据实体来构造下拉框数据
getChooseMap(Class<?> clz, String key, String value)	根据实体来构造下拉框数据，有 headerValue
getChooseMap(Enum<?>[] enums)	根据枚举来构造下拉框数据，有 headerValue

✓ LoggerUtils

名称	功能
getLogger(Class<?> clz)	获取日志类

✓ MessageUtils

用于向页面请求返回 JSON 格式的消息。

名称	功能
success(String msg)	返回成功消息
failure(Exception e, Class<?> clazz)	返回失败消息 ,clazz 为最终捕获异常的类 ,也就是 failure() 的使用者 ,作为日志记录者来记录普通异常和 service 层无法捕获的 CannotCreateTransactionException 异常。

✓ I18nUtils

从资源文件获取国际化信息。

名称	功能
getMessage(String parameter)	获取国际化信息 , 如果不存在则直接返回 parameter

✓ WebUtils

用于获取 request , response , session 和直接向页面输出不同格式的数据。

名称	功能
getRequest()	获取 request
getParameter(String name)	获取 request 中的参数值
getResponse()	获取 response
getSession()	获取 session
getSession(boolean isNew)	获取 session
getSessionAttribute(String name)	获取 session 的属性值
renderText()	输出文本
renderHtml()	输出 html
renderXml()	输出 xml
renderJson()	输出 json
renderJsonp()	跨域输出 json
setDisableCacheHeader(HttpServletResponse response)	设置禁止客户端缓存的 Header

✓ EasyUiUtils

用于把不同的对象转换成 EasyUiDataGrid。

名称	功能
getEasyUiDataGrid(Page pageEntity)	把 page 转换成 EasyUiDataGrid
getEasyUiDataGrid(List<?> list)	把 list 转换成 EasyUiDataGrid
getEasyUiDataGrid(List<?> list, int total)	把 list 转换成 EasyUiDataGrid 并指定总数

✓ EncodeUtils

加密工具类。

名称	功能
toMD5(String pwd)	MD5 加密
unescape(String src)	对应 javascript 的 unescape()函数, 可对 javascript 的 escape()进行解码

✓ GzipUtils

解压缩文件工具类。

名称	功能
zipFile(File file, File zipFile)	压缩文件
unZipFile(InputStream is, File zipFile)	解压文件

✓ CoreUtils

CoreUtils 定义一些常用方法，里面还有一个 spring 的上下文的静态变量 *context*。

名称	功能
executeSql(String sqlPath, ApplicationContext context)	执行 sql 文件
code2value	用来转义
object2int(Object o)	把通过 sql 查询出来的 BigDecimal , Long , Integer 转成 Integer 类型
convertStringToObject(String value, Class<Y> toType)	强制把 String 类型转成对应的类型
validateEntity(Object entity)	校验实体
handleException(String message,Exception e,Logger logger)	处理异常

✓ PropertyUtils

配置文件读取工具类

名称	功能
get(Object key)	读取配置参数，默认是读取 application.properties
get(Object key, String file)	指定配置文件读取参数值

3.8 JavaScript 介绍

对于一个基于 Ajax 调用的开发框架来说，JS 脚本是一个重要的组成部分，本章主要介绍两个内容：开发框架的 JS 工具类和 JS 校验。

1) JS 工具类

✓ syUtils

syUtils 位于\src\main\webapp\resources\js\syUtil.js，主要方法包括：

名称	功能
sy.fs()	formatString 功能。
sy.ns()	增加命名空间功能。
sy.ajaxSubmit(form, window, datagrid)	异步表单提交。
sy.postDel(ids,url,datagrid)	删除数据
sy.batchDel(datagrid,delMethod)	批量删除数据
sy.search(datagridId,serarchFormId)	按条件过滤数据表格
sy.datagrid(id,datagrid,columns,isFrozen,operation)	构造数据表格
sy.trim()	过滤表单元素的前后空白字符串
sy.bp()	获得项目根路径
sy.showLoadingDiv()	显示 AJAX 开始时的提示信息
sy.hideLoadingDiv()	AJAX 结束时隐藏提示信息
sy.dataGridHeadCheckBoxUnSelect(dataGrid)	修正 jqueryEasyUi 的一个 BUG，取消复选框的选中，在数据表格的 onLoadSuccess 里调用。
sy.UUID()	生成 UUID
sy.buildAddButton(buttons,permission,win,winParam)	生成增加功能按钮
sy.buildDelButton (buttons,permission,datagird,delFunction,module)	生成删除功能按钮

sy.buildQueryButton(buttons,permission,win,winParam)	生成查询功能按钮
sy.buildExportButton (buttons,permission,searchForm,url)	生成导出功能按钮

2) JS 校验

虽然开发框架提供了实体的校验，但在前端的 JS 校验同样也不可或缺，理由为用户提交表单前进行 JS 校验不单比后端体验更友好，而且也减轻了服务器的负载。Easyui 的校验框架功能全面，容易使用和扩展。下面对 Easyui 的校验框架的使用进行简单介绍。首先，新增学生时对年龄进行校验，修改 student_input.jsp 的输入框代码，如图：

```
<input name="age" value="${age}" class="easyui-numberbox" required="true" min="6" max="20"/>
```

(图 3.8-1)

对该输入增加 class 属性，“easyui-numberbox”标识这是一个数字输入框，所有非数字的输入都会被过滤，required="true"标识这是个必要元素，如果没有输入则提示“该输入项为必填项”，在 input 标签增加 missingMessage="提示信息"更改默认提示信息，min，max 为最小最大值。然后我们对学生名字的长度进行校验，代码如下：

```
<input name="name" value="${name}" class="easyui-validatebox" required="true" validType="length[2,50]"/>
```

(图 3.8-2)

这里使用的是“easyui-validatebox”，需要指定 validType，length 为长度校验规则，[2,50] 为校验参数，分别代表最小长度和最大长度，validType 可支持多规则同时校验，例如 validType="length[10,50]&email"。在 input 标签增加 invalidMessage="提示信息"更改校验失败时的默认提示信息。接着介绍校验规则的扩展，假设有这样的需求，更改密码时必须和原密码不一样。首先扩展一个新旧密码不同的规则，在 student_input.js 增加如下代码：

```
$.extend($.fn.validatebox.defaults.rules, {
    differentPwd: {
        validator: function(value, param){
            return value==$(param[0]).val()?false:true;
        },
        message: "新旧密码不能相同"
    }
});
```

(图 3.8-3)

在 student_input.jsp 增加如下代码：

```
<input id="demo_student_pwd" name="pwd" type="hidden" value="abc123"/>
<input name="pwd1" class="easyui-validatebox" required="true" validType="differentPwd['#demo_student_pwd']"/>
```

(图 3.8-4)

旧密码的 input 标签的 id 为 demo_student_pwd , 并作为参数传到校验规则里面 , 所以校验规则用 value==\$(param[0]).val() 来判断新旧密码是否相同。校验参数还可以把 EL 表达式的值传进去 , 例如 differentPwd['\${pwd}'] , 运行结果如图：

(图 3.8-5)

如果校验规则会被重复使用，请把扩展代码写在

\src\main\webapp\resources\js\syValidateRule.js 里面。想进一步了解 easyui 校验可以

访问：<http://www.jeasyui.com/>

3.9 国际化

国际化 (Internationalization) 是设计一个适用于多种语言和地区的应用程序的过程 , 国际化有时候被简称为 i18n , 因为有 18 个字母在国际化的英文单词的字母 i 和 n 之间。我们面对的主要是服务器端和前端脚本的国际化问题。

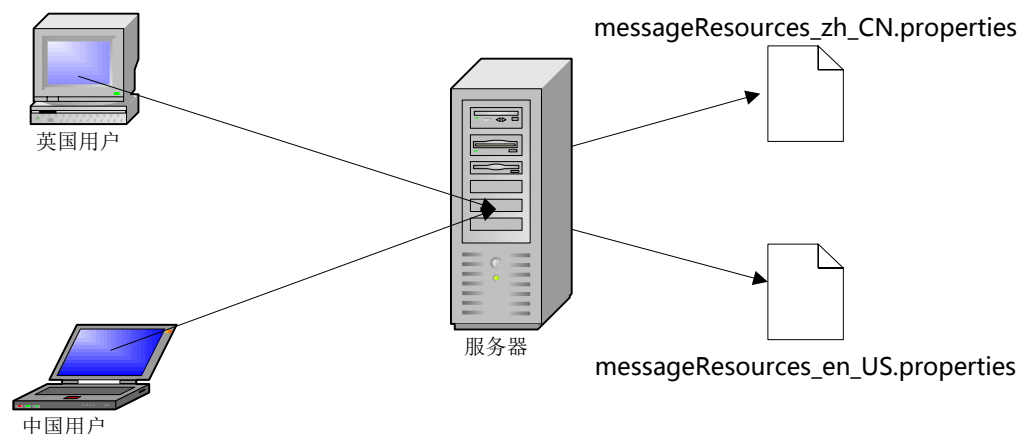
1) 服务器端国际化

服务器端的国际化底层是依赖于 struts2 的实现 , 首先 struts.xml 要对国际化文件进行声明 , 如图 :

```
<!--国际化-->
<constant name="struts.custom.i18n.resources" value="messageResources"></constant>
```

(图 3.9-1)

另外还需要在 struts.xml 的同级目录下增加对应的地区的资源文件 , 以在 struts.xml 中定义的 messageResources 为前缀 , 中文地区的是 messageResources_zh_CN.properties , 英文地区的是 messageResources_en_US.properties , 这样一来 , struts 就可以根据用户所在的地区进行语言的本地化了 , 如图 :



(图 3.9-2)

服务端的国际化又细分几个类型 :

- ✧ 页面 JSP , 使用 <s:text> 标签 , name 属性对应对应资源文件的 key 值。
- ✧ 枚举类 , 在 getDesc 方法里调用工具类 I18nUtils.getMessage(desc) , desc 参数对应资源

文件的 key 值。

- ✧ 业务异常，构造方法 `ServiceException(String message)` 的 `message` 参数对应资源文件的 key 值。
- ✧ 实体校验信息，注解 `Validation` 的 `name` 属性，`regex` 属性中的提示信息对应资源文件的 key 值。

枚举类，业务异常，实体校验信息底层都是使用了工具类 `l18nUtils`，如果在资源文件没有找到对应的值则直接返回 key 值。

2) 前端脚本国际化

前端脚本国际化的原理就是把固定的文字转换成 javascript 变量放在独立的脚本文件里，不同的语言有不同的脚本，路径在 `\src\main\webapp\resources\js\language`，例如学生管理提供国际化支持，在该路径下增加 `student` 目录及 `student_zh.js`，`student_en.js`，在 jsp 页面引入 `student_{language}.js` 就会引入对应语言的脚本。脚本的国际化一般分为几个类型：

- ✧ 按钮文字
- ✧ 弹出窗口标题
- ✧ 提示文字
- ✧ 数据表格标题
- ✧ 校验信息

3.10 配置文件

系统的配置文件位于项目的 `src/main/resources` 目录。

- ✧ `application.properties`，配置数据库连接，连接池等信息，其中 `app.id` 指向权限表的系统 id。
- ✧ `applicationContext-repository.xml`，`spring-data-jpa` 配置文件，主要是指定 `dao` 所在的包

路径。

- ✧ applicationContext-security.xml，安全框架配置文件。
- ✧ applicationContext.xml，spring 主配置文件，要根据数据库类型指定 jdbcDialect 和 codeJdbcDao 的实现类，另外 jpaCorePointcut 是框架核心 service 的事务，根据项目的不同，指定 jpaBizPointcut 的包路径，让该路径的 service 类都有事务保障。
- ✧ messageResources_*.properties，国际化资源文件。
- ✧ struts.xml，struts2 配置文件，主要指定 action 层所在包路径。

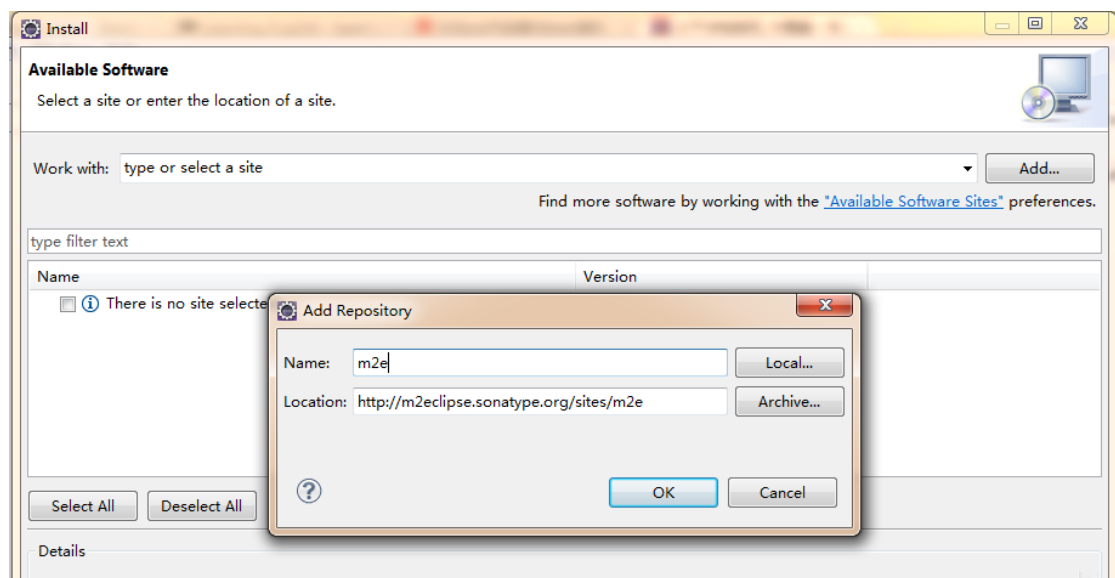
4. 其它

4.1 开发环境

4.1.1 安装 maven 插件

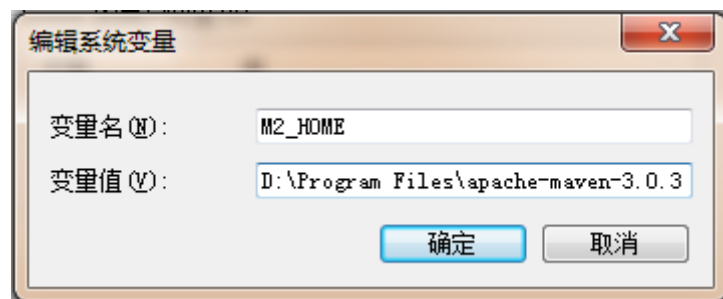
通过 eclipse 的 Help 菜单安装，分别点击 Help—Install New Software...，弹出安装界面后点击 Add 按钮，在弹出框的 name 项输入 m2e，location 项输入

<http://m2eclipse.sonatype.org/sites/m2e>，如图：



(图 4.1-1)

选中后执行安装。接着在 <http://maven.apache.org/download.html> 下载 maven，版本 3.0 以上，本例 maven 放在本地 D:\Program Files，设置系统环境变量 M2_HOME，如图：



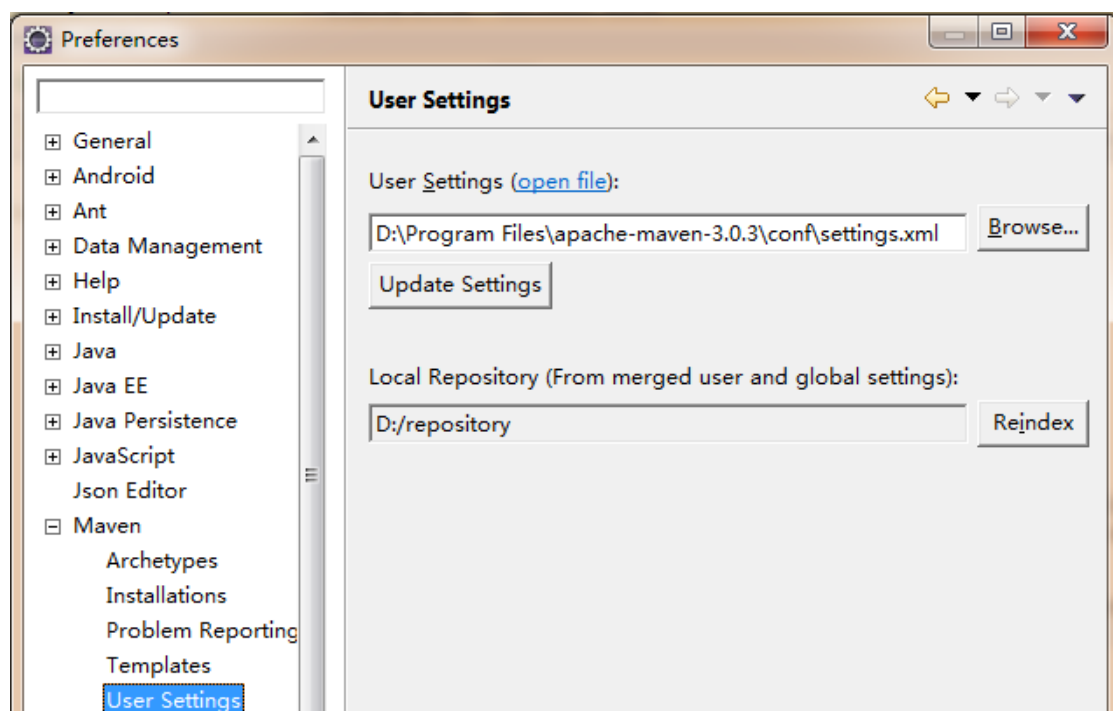
(图 4.1-2)

在 M2_HOME\conf 文件夹下修改 setting 文件，修改<localRepository>标签的内容，指向本地仓库（任意文件夹都可以），然后修改远程服务器地址，如图：

```
<mirror>
  <id>nexus</id>
  <url>http://192.168.2.102:8081/nexus/content/groups/public</url>
  <mirrorOf>*</mirrorOf>
</mirror>
```

(图 4.1-3)

打开 eclipse 的 Window—Preference—Maven—User Settings 界面，设置 maven 的 setting 文件路径，如图：

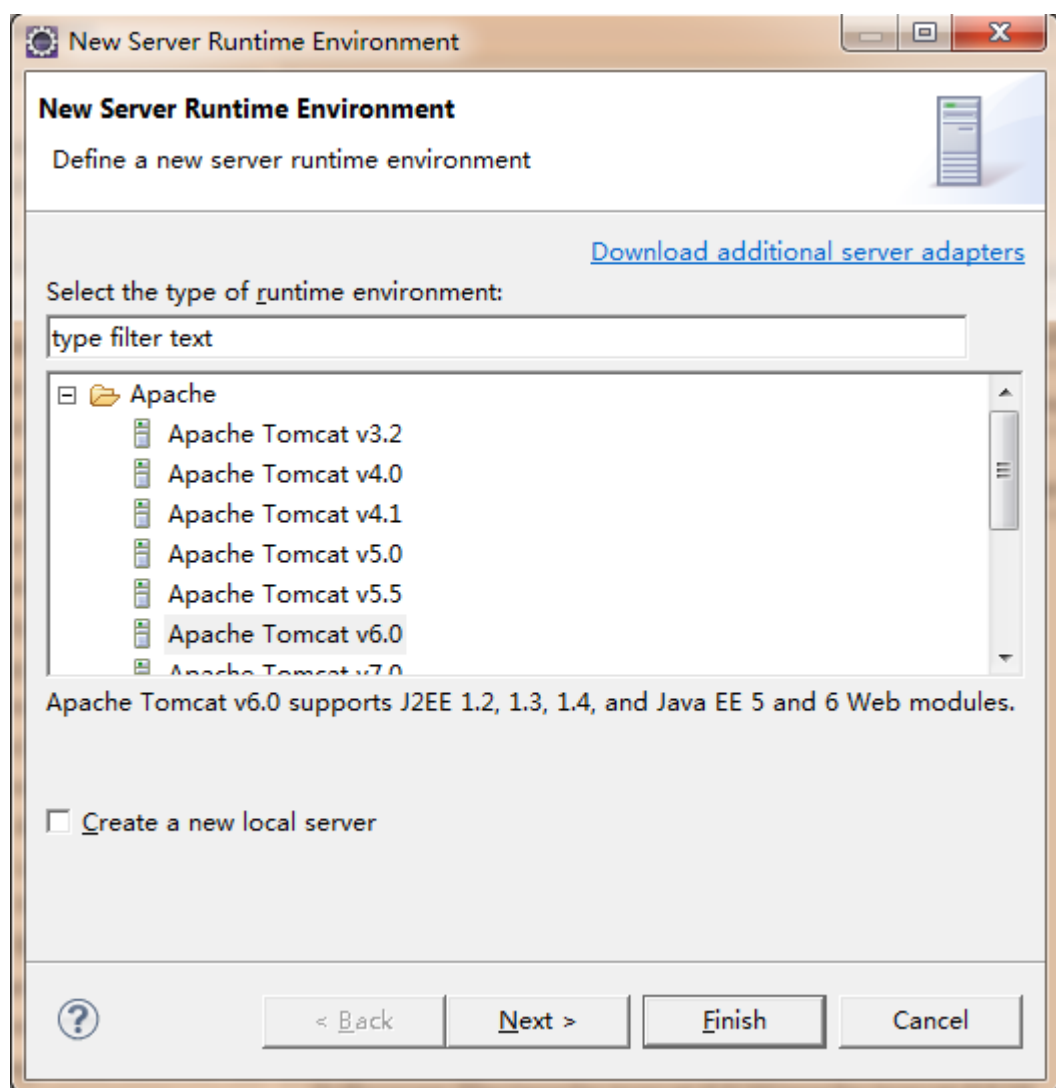


(图 4.1-4)

Maven 开发环境至此已经配置完，导入 maven 工程就可以开始我们的开发了。

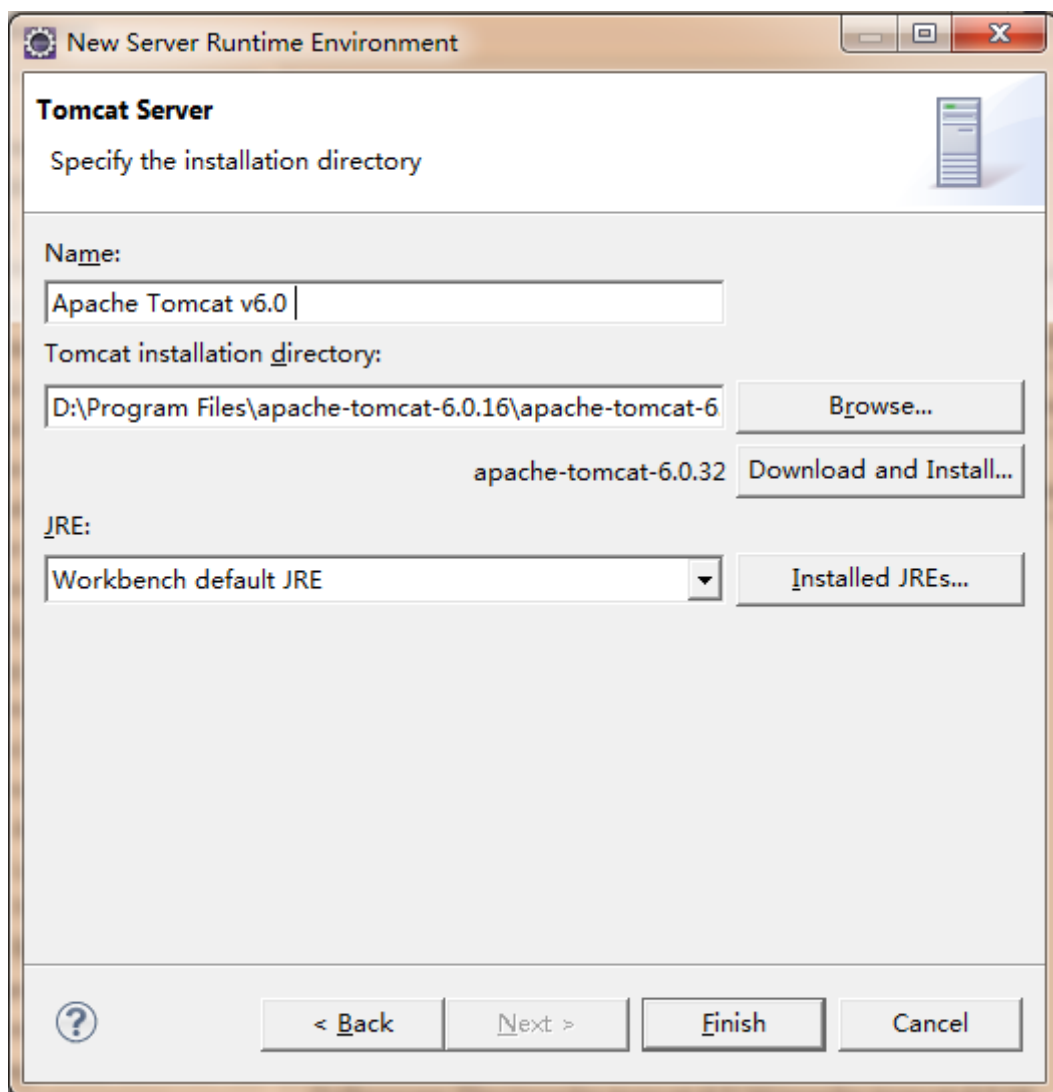
4.1.2 配置 tomcat

从 <http://tomcat.apache.org/download-60.cgi> 下载 tomcat6 到本地任意文件夹，然后配置 eclipse 的 tomcat 运行环境，打开 Window—Preference—Server—Run time environments，点击 Add 按钮，弹出以下界面：



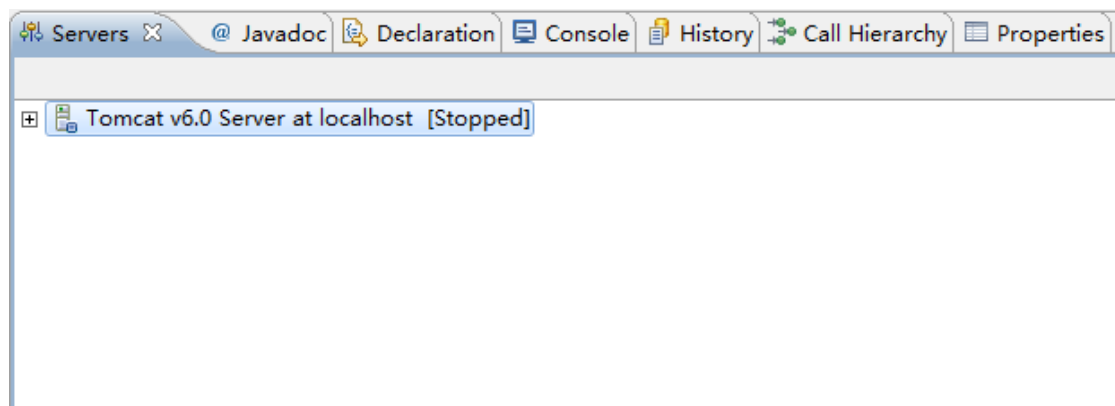
(图 4.1-5)

选择 Apache Tomcat v6.0，点击 Next 按钮，在 Tomcat install directory 指定 tomcat 的本地路径，如图：



(图 4.1-6)

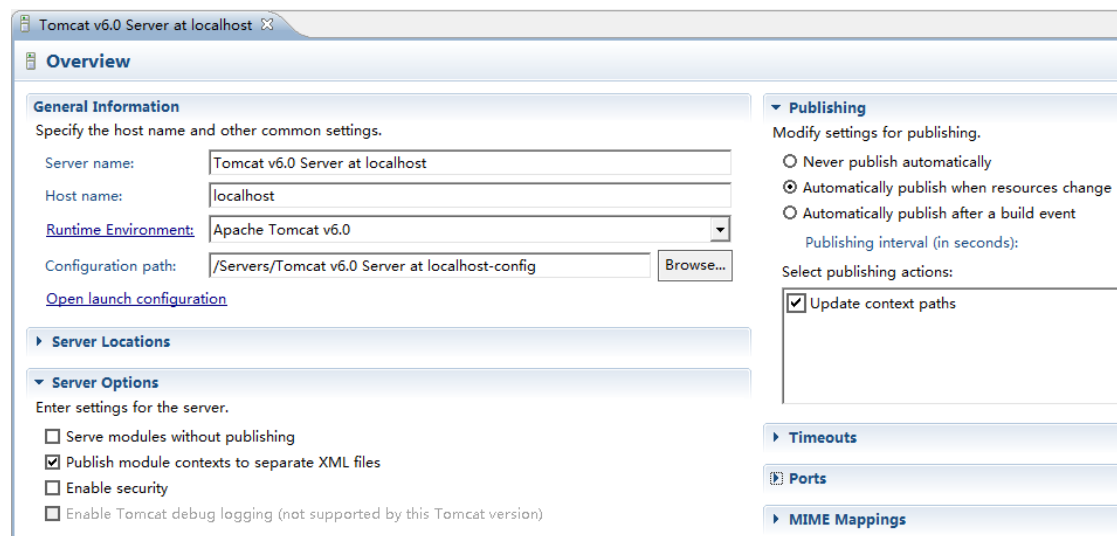
打开 Servers 视图并新建一个服务器，如图：



(图 4.1-7)

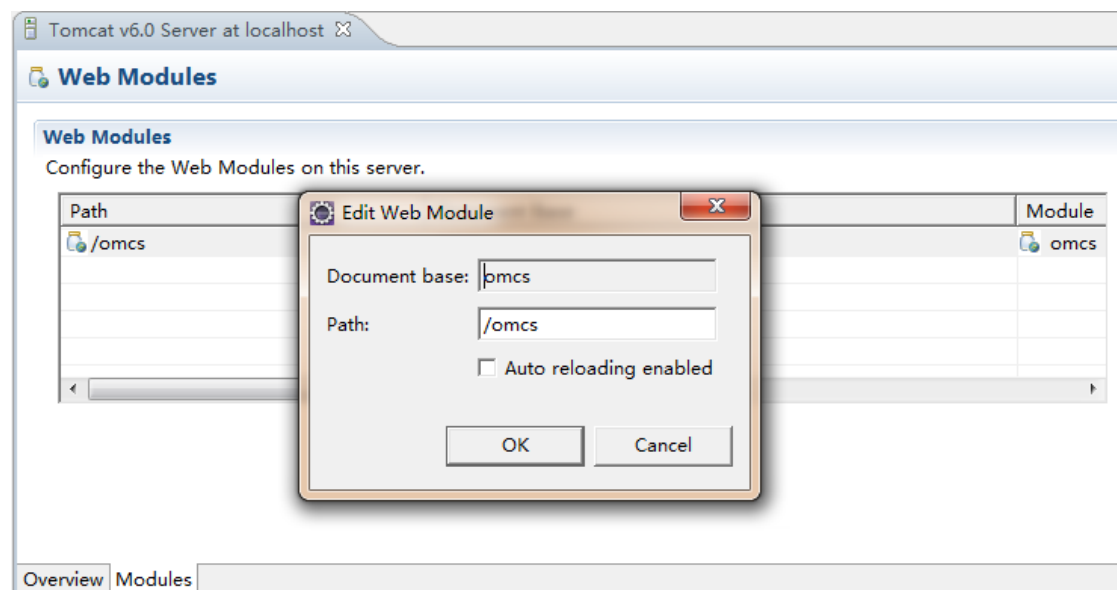
在 tomcat 加入 maven 工程后，要配置 tomcat 的热部署，这样可以减少调试代码过程中的 tomcat 重启次数。双击刚刚新建的服务器 确保“Publish module contexts to separate

XML files” 和 “Automatically publish when resources change” 两项被选中，如图：



(图 4.1-8)

切换到 Modules，把自动加载设为 false，如图：




(图 4.1-9)

现在做最后一步配置，右击 maven 工程—Properties—Deployments Assembly，把 maven 依赖添加进来，这样我们定义在 pom 文件的依赖 jar 包才能发布到 tomcat 服务器上，如图：



(图 4.1-10)



提示：

当启动 tomcat 提示找不到类时，第一我们要确认 pom 文件没有报错，所有的 jar 包都在本地仓库；第二就是 maven 依赖已经添加到 Deployments Assembly；确认以上两个都没问题时，右键 tomcat 服务器—clean，重新发布工程。

4.2 编码规范

编码需要规范化的原因有不少，但是最重要的一个就是提高程序的可读性，让项目后期的升级和维护更简单。

4.2.1 Java 规范

- ✧ 类名以大写字母开头，单词首字母大写。
- ✧ 方法以动词开头。
- ✧ 普通变量使用驼峰命名，静态变量要大写字母，单词间用“_”分隔。
- ✧ 不要使用汉语拼音。
- ✧ 使用 eclipse 的格式化功能对代码进行格式化。
- ✧ 对 service 层的 public 方法进行注释。
- ✧ 关键逻辑要写注释。

4.2.2 JS 脚本规范

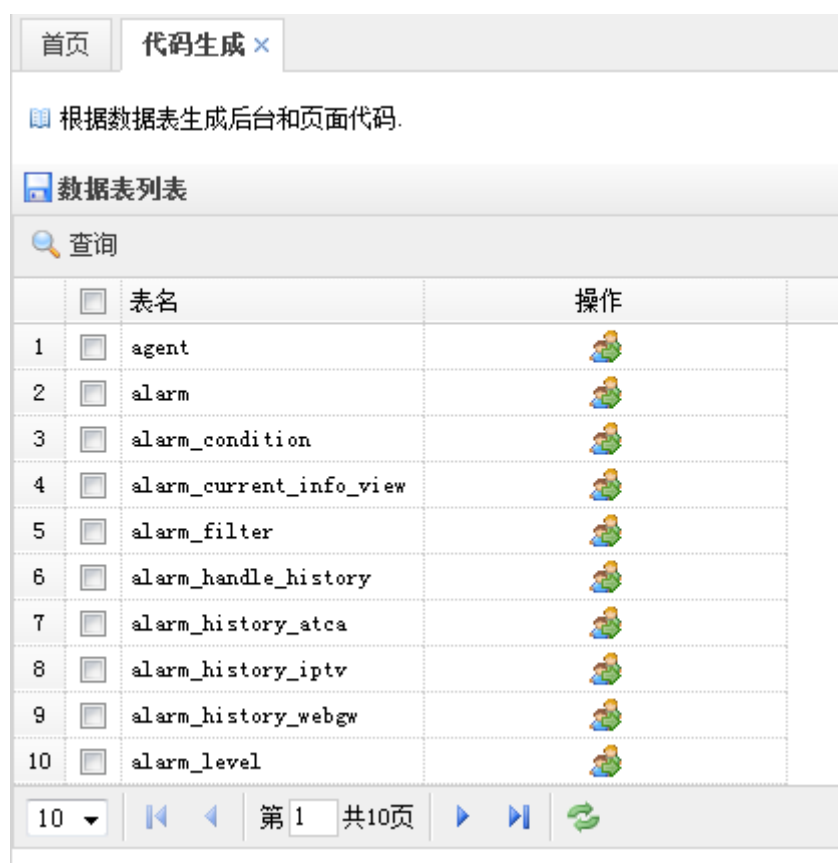
- ✧ 脚本从 html 或 jsp 分离出去，以小写字母开头。
- ✧ 变量命名与 Java 代码一致。
- ✧ 尽量使用 jQuery。
- ✧ 动态加载的脚本，例如 classinfo.js，classinfo_datagrid.js 等脚本，定义的方法和全局变量要使用命名空间，格式为“模块名.功能名.方法名或变量名”。

4.2.3 Html 规范

- ✧ 标签一般都要闭合。
- ✧ 标签间要注意换行和缩进，提高可读性。
- ✧ 如果标签有 id 属性，命名一定要符合“模块名_功能名_对象名”，这样打开多个页面时也不会 id 冲突，通过脚本获取的值也是唯一的。

4.3 代码生成

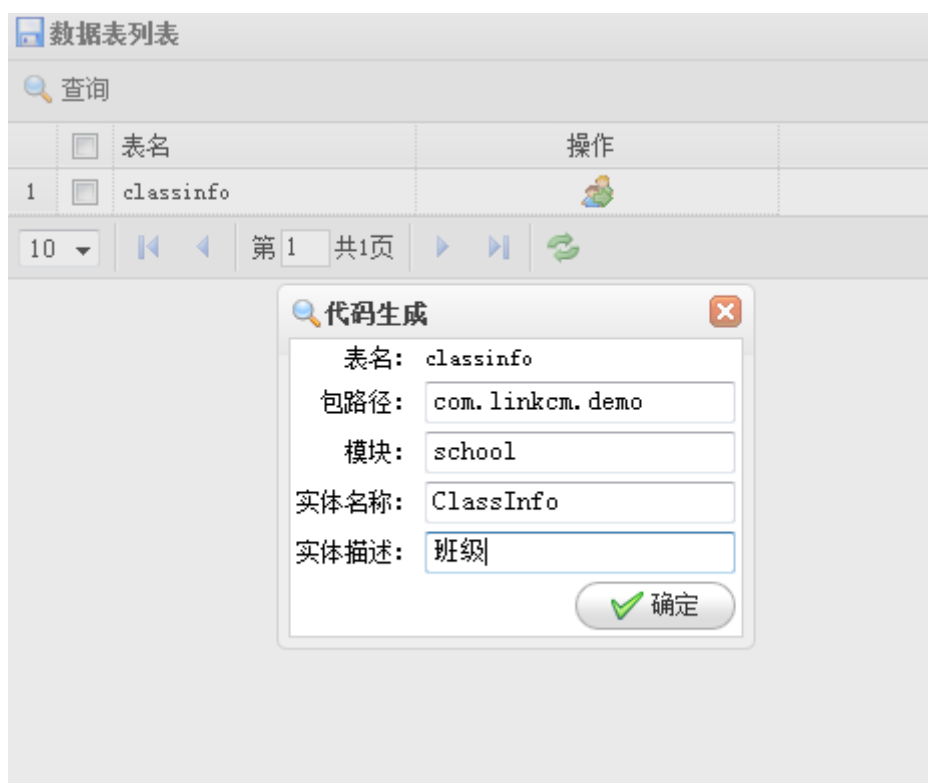
在常规的功能开发中，我们要实现一个模块功能，代码上首先写好实体，枚举类（假设有）DAO，Service，Action，接着就是相关页面，总体都是这个步骤，虽然也有工具提供了根据数据库信息来生成实体，但是没法生成和开发框架有关的其它代码，例如 Service，Action，页面。如果可能通过数据库信息来生成通用部分代码，程序员只需要增加或修改部分代码，那开发效率就能获得大大的提高，我们的开发框架正好提供了这个功能。运行界面如图：



(图 4.3-1)

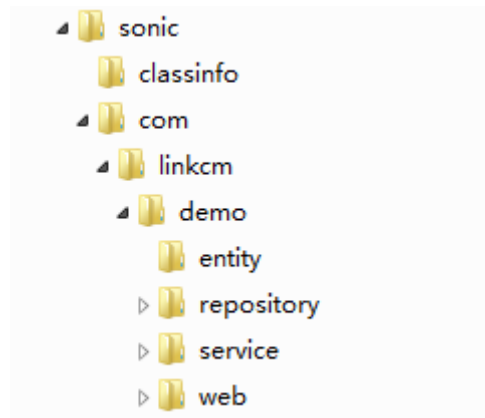
通过查询功能找到需要生成代码的数据表, 点击操作功能按钮, 弹出对话框, 输入有关信息,

如图:



(图 4.3 - 2)

表单里的 4 个字段都是必填的，分别是包路径--代表了包路径的前缀；模块名--代码的完整路径包括了包路径+模块名，JS 脚本和页面生成在与该名字相同的文件夹下；实体名--实体名称，也是 Dao，Service，Action 等 java 类的前缀；描述--对实体的描述，提供给记录日志或者提示信息功能；生成的代码位于 D:\sonic 目录，结果如图：



(图 4.3- 3)

数据表的信息尽量详细，例如字段长度，是否非空，唯一索引，这些信息生成到实体类或服务类里面，省去手工增加的麻烦。假设要生成枚举类，数据表的 D D L 对应字段的注释要以下规则，例如学生表的性别字段：

```
CREATE TABLE `student` (  
  `student_id` bigint(20) NOT NULL AUTO_INCREMENT,  
  `age` int(11) DEFAULT NULL COMMENT '年龄',  
  `class_id` bigint(20) DEFAULT NULL COMMENT '班级',  
  `name` varchar(255) DEFAULT NULL COMMENT '姓名',  
  `sex` int(11) DEFAULT NULL COMMENT '性别'  
  0: 男  
  1: 女,  
  PRIMARY KEY (`student_id`)  
)
```

(图 4.3- 4)

首先对 “sex” 字段的描述是 “性别”，然后换行，类别男女之间也要换行，名值对以冒号分隔，结果如图：

```

public enum StudentSex {

    ♂(0),
    ♀(1);

    public final int value;

    public static final Transformer transformer = new AbstractTransformer() {
        public String getDesc(Object type) {
            return StudentSex.getDesc(CoreUtils.object2int(type));
        }
    };

    private StudentSex(int value) {
        this.value = value;
    }

    public String getDesc() {
        return I18nUtils.getMessage(name());
    }
}

```

(图 4.3- 5)