

学成在线 第3天 讲义-CMS页面管理开发

1 自定义条件查询

1.1 需求分析

在页面输入查询条件，查询符合条件的页面信息。

查询条件如下：

站点Id：精确匹配

模板Id：精确匹配

页面别名：模糊匹配

...

1.2 服务端

1.2.1 Dao

使用 CmsPageRepository中的findAll(Example<S> var1, Pageable var2)方法实现，无需定义。

下边测试findAll方法实现自定义条件查询：

```
//自定义条件查询测试
@Test
public void testFindAll() {
    //条件匹配器
    ExampleMatcher exampleMatcher = ExampleMatcher.matching();
    exampleMatcher = exampleMatcher.withMatcher("pageAliase",
ExampleMatcher.GenericPropertyMatchers.contains());
    //页面别名模糊查询，需要自定义字符串的匹配器实现模糊查询
    //ExampleMatcher.GenericPropertyMatchers.contains() 包含
    //ExampleMatcher.GenericPropertyMatchers.startsWith()//开头匹配
    //条件值
    CmsPage cmsPage = new CmsPage();
    //站点ID
    cmsPage.setSiteId("5a751fab6abb5044e0d19ea1");
    //模板ID
    cmsPage.setTemplateId("5a962c16b00ffc514038fafd");
    //
    cmsPage.setPageAliase("分类导航");
    //创建条件实例

    Example<CmsPage> example = Example.of(cmsPage, exampleMatcher);
```

```
Pageable pageable = new PageRequest(0, 10);
Page<CmsPage> all = cmsPageRepository.findAll(example, pageable);
System.out.println(all);
}
```

1.2.2 Service

在PageService的findlist方法中增加自定义条件查询代码

```
/**
 * 页面列表分页查询
 * @param page 当前页码
 * @param size 页面显示个数
 * @param queryPageRequest 查询条件
 * @return 页面列表
 */
public QueryResponseResult findList(int page,int size,QueryPageRequest queryPageRequest){
    //条件匹配器
    //页面名称模糊查询，需要自定义字符串的匹配器实现模糊查询
    ExampleMatcher exampleMatcher = ExampleMatcher.matching()
        .withMatcher("pageAliase", ExampleMatcher.GenericPropertyMatchers.contains());
    //条件值
    CmsPage cmsPage = new CmsPage();
    //站点ID
    if(StringUtils.isNotEmpty(queryPageRequest.getSiteId())){
        cmsPage.setSiteId(queryPageRequest.getSiteId());
    }
    //页面别名
    if(StringUtils.isNotEmpty(queryPageRequest.getPageAliase())){
        cmsPage.setPageAliase(queryPageRequest.getPageAliase());
    }

    //创建条件实例
    Example<CmsPage> example = Example.of(cmsPage, exampleMatcher);
    //页码
    page = page-1;
    //分页对象
    Pageable pageable = new PageRequest(page, size);
    //分页查询
    Page<CmsPage> all = cmsPageRepository.findAll(example,pageable);
    QueryResult<CmsPage> cmsPageQueryResult = new QueryResult<CmsPage>();
    cmsPageQueryResult.setList(all.getContent());
    cmsPageQueryResult.setTotal(all.getTotalElements());
    //返回结果
    return new QueryResponseResult(CommonCode.SUCCESS,cmsPageQueryResult);
}
```

1.2.3 Controller

无需修改

1.2.4 测试

使用SwaggerUI测试

Parameters

Parameter	Value	Description	Parameter Type	Data Type
siteId	<input type="text" value="5a751fab6abb5044e0d19ea1"/>	站点id	query	string
pageId	<input type="text"/>		query	string
pageName	<input type="text"/>		query	string
pageAliase	<input type="text" value="轮播图"/>		query	string
templateId	<input type="text"/>		query	string
page	<input type="text" value="1"/>	页码	path	integer
size	<input type="text" value="11"/>	每页记录数	path	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

Try it out!

[Hide Response](#)

1.3 前端

1.3.1 页面

1、增加查询表单

```
<!-- 查询表单 -->
<el-form :model="params">
  <el-select v-model="params.siteId" placeholder="请选择站点">
    <el-option
      v-for="item in siteList"
      :key="item.siteId"
      :label="item.siteName"
      :value="item.siteId">
    </el-option>
  </el-select>
  页面别名: <el-input v-model="params.pageAliase" style="width: 100px;"></el-input>
  <el-button type="primary" v-on:click="query" size="small">查询</el-button>
</el-form>
```

2、数据模型对象

增加siteList、pageAliase、siteId，如下：

```
data() {
```

```
return {
  siteList:[],//站点列表
  list:[],
  total:50,
  params:{
    siteId:'',
    pageAliase:'',
    page:1,//页码
    size:2//每页显示个数
  }
}
```

3、在钩子方法中 构建siteList站点列表

```
mounted() {
  //默认查询页面
  this.query()
  //初始化站点列表
  this.siteList = [
    {
      siteId:'5a751fab6abb5044e0d19ea1',
      siteName:'门户主站'
    },
    {
      siteId:'102',
      siteName:'测试站'
    }
  ]
}
```

1.3.2 Api调用

1、向服务端传递查询条件，修改 cms.js，如下：

```
//public是对axios的工具类封装，定义了http请求方法
import http from '../.../base/api/public'
import querystring from 'querystring'
let sysConfig = require('@../config/sysConfig')
let apiUrl = sysConfig.xcApiUrlPre;

export const page_list = (page,size,params) => {
  //将json对象转成key/value对
  let query = querystring.stringify(params)
  return http.requestQuickGet(apiUrl+'/cms/page/list/'+page+'/'+size+'/?'+query)
}
```

2、页面调用api方法

```
//查询
query:function () {
    cmsApi.page_list(this.params.page,this.params.size,this.params).then((res)=>{
        console.log(res)
        this.total = res.queryResult.total
        this.list = res.queryResult.list
    })
}
```

测试如下：



2 新增页面

2.1 新增页面接口定义

1、定义响应模型

```
@Data
public class CmsPageResult extends ResponseResult {
    CmsPage cmsPage;
    public CmsPageResult(ResultCode resultCode,CmsPage cmsPage) {
        super(resultCode);
        this.cmsPage = cmsPage;
    }
}
```

2、定义添加Api

在api工程中添加接口：

```
@ApiOperation("添加页面")
public CmsPageResult add(CmsPage cmsPage);
```

2.2 新增页面服务端开发

2.2.1 页面唯一索引

在cms_page集中上创建页面名称、站点Id、页面webpath为唯一索引

2.2.2 Dao

1、添加根据页面名称、站点Id、页面webpath查询页面方法，此方法用于校验页面是否存在

```
public interface CmsPageRepository extends MongoRepository<CmsPage,String> {  
    //根据页面名称、站点id、页面访问路径查询  
    CmsPage findByPageNameAndSiteIdAndPageWebPath(String pageName,String siteId,String  
    pageWebPath);  
    . . .
```

2、使用 CmsPageRepository提供的save方法。

2.2.3 Service

```
//添加页面  
public CmsPageResult add(CmsPage cmsPage){  
    //校验页面是否存在，根据页面名称、站点Id、页面webpath查询  
    CmsPage cmsPage1 =  
    cmsPageRepository.findByPageNameAndSiteIdAndPageWebPath(cmsPage.getPageName(),  
    cmsPage.getSiteId(), cmsPage.getPageWebPath());  
    if(cmsPage1==null){  
        cmsPage.setPageId(null);//添加页面主键由spring data 自动生成  
        cmsPageRepository.save(cmsPage);  
        //返回结果  
        CmsPageResult cmsPageResult = new CmsPageResult(CommonCode.SUCCESS,cmsPage);  
        return cmsPageResult;  
    }  
    return new CmsPageResult(CommonCode.FAIL,null);  
}
```

2.2.4 Controller

```
//添加页面  
@Override  
@PostMapping("/add")  
public CmsPageResult add(@RequestBody CmsPage cmsPage) {  
    return pageService.add(cmsPage);  
}
```

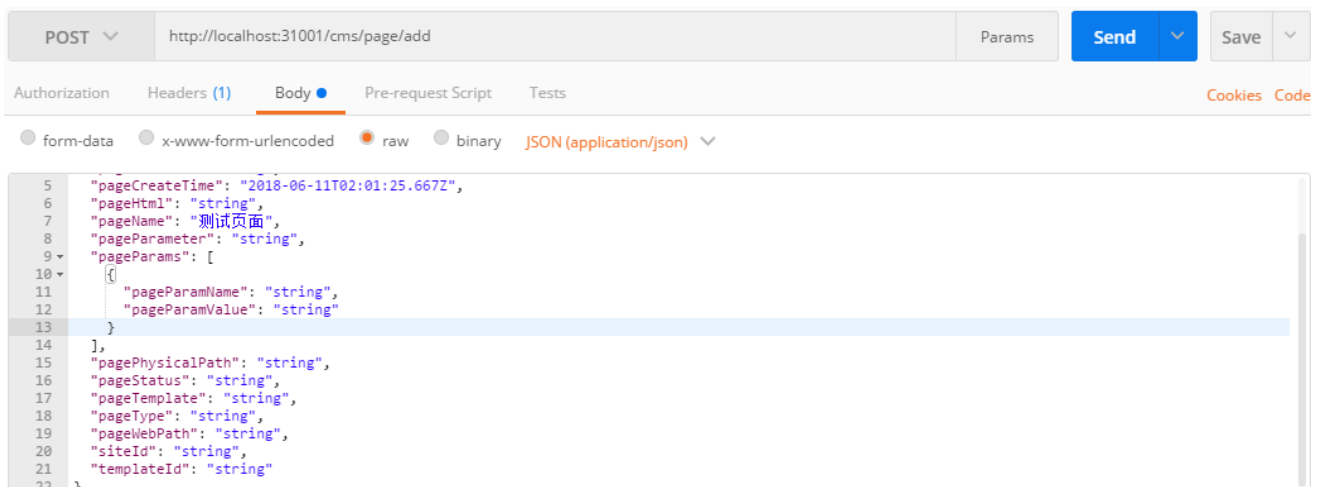
2.2.5 接口测试

使用postman测试

post请求：<http://localhost:31001/cms/page/add>

请求内容为json数据，测试数据如下：

```
{
  "dataUrl": "string",
  "htmlFileId": "string",
  "pageAliase": "string",
  "pageCreateTime": "2018-06-11T02:01:25.667Z",
  "pageHtml": "string",
  "pageName": "测试页面",
  "pageParameter": "string",
  "pagePhysicalPath": "string",
  "pageStatus": "string",
  "pageTemplate": "string",
  "pageType": "string",
  "pageWebPath": "string",
  "siteId": "string",
  "templateId": "string"
}
```



成功响应结果：

```
Pretty Raw Preview JSON ↕
1 {
2   "success": true,
3   "code": 10000,
4   "message": "操作成功!",
5   "cmsPage": {
6     "siteId": "string",
7     "pageId": "5b1dd8e3319a902dc0843e7d",
8     "pageName": "测试页面",
9     "pageAlias": "string",
10    "pageWebPath": "string",
11    "pageParameter": "string",
12    "pagePhysicalPath": "string",
13    "pageType": "string",
14    "pageTemplate": "string",
15    "pageHtml": "string",
16    "pageStatus": "string",
17    "pageCreateTime": 1528682485667,
18    "templateId": "string",
19    "pageParams": [
20      {
21        "pageParamName": "string",
22        "pageParamValue": "string"
23      }
24    ],
25    "htmlFileId": "string",
26    "dataUrl": "string"
27  }
28 }
```

失败响应结果：

```
Pretty Raw Preview JSON ↕
1 {
2   "success": false,
3   "code": 11111,
4   "message": "操作失败!",
5   "cmsPage": null
6 }
```

2.3 新增页面前端开发

2.3.1 新增页面

2.3.1.1 编写page_add.vue页面

使用Element-UI的form组件编写添加表单内容，页面效果如下：

添加页面

* 所属站点

* 选择模版

* 页面名称

别名

* 访问路径

* 物理路径

类型 ☐ 静态 ☐ 动态

创建时间

1、创建page_add.vue页面

2、配置路由

在cms模块的路由文件中配置“添加页面”的路由：

```
{path: '/cms/page/add', name: '新增页面', component: page_add, hidden: true}
```

注意：由于“添加页面”不需要显示为一个菜单，这里hidden设置为true隐藏菜单。

测试，在浏览器地址栏输入<http://localhost:11000/#/cms/page/add>

3、在页面列表添加“添加页面”的按钮

实际情况是用户进入页面查询列表，点击“新增页面”按钮进入新增页面窗口。

在查询按钮的旁边添加：

```
<router-link class="mui-tab-item" :to="{path: '/cms/page/add/'}">
  <el-button type="primary" size="small">新增页面</el-button>
</router-link>
```

说明：router-link是vue提供的路由功能，用于在页面生成路由链接，最终在html渲染后就是<a标签。

to：目标路由地址

4、完善页面内容：

代码如下：

```
<el-form :model="pageForm" label-width="80px" >
  <el-form-item label="所属站点" prop="siteId">
    <el-select v-model="pageForm.siteId" placeholder="请选择站点">
      <el-option
```



```
      v-for="item in siteList"
      :key="item.siteId"
      :label="item.siteName"
      :value="item.siteId">
    </el-option>
  </el-select>
</el-form-item>
<el-form-item label="选择模版" prop="templateId">
  <el-select v-model="pageForm.templateId" placeholder="请选择">
    <el-option
      v-for="item in templateList"
      :key="item.templateId"
      :label="item.templateName"
      :value="item.templateId">
    </el-option>
  </el-select>
</el-form-item>
<el-form-item label="页面名称" prop="pageName">
  <el-input v-model="pageForm.pageName" auto-complete="off" ></el-input>
</el-form-item>

<el-form-item label="别名" prop="pageAliase">
  <el-input v-model="pageForm.pageAliase" auto-complete="off" ></el-input>
</el-form-item>
<el-form-item label="访问路径" prop="pageWebPath">
  <el-input v-model="pageForm.pageWebPath" auto-complete="off" ></el-input>
</el-form-item>

<el-form-item label="物理路径" prop="pagePhysicalPath">
  <el-input v-model="pageForm.pagePhysicalPath" auto-complete="off" ></el-input>
</el-form-item>

<el-form-item label="类型">
  <el-radio-group v-model="pageForm.pageType">
    <el-radio class="radio" label="0">静态</el-radio>
    <el-radio class="radio" label="1">动态</el-radio>
  </el-radio-group>
</el-form-item>
<el-form-item label="创建时间">
  <el-date-picker type="datetime" placeholder="创建时间" v-model="pageForm.pageCreateTime">
</el-date-picker>
</el-form-item>

</el-form>
<div slot="footer" class="dialog-footer">
  <el-button type="primary" @click="addSubmit" >提交</el-button>
</div>
```

Form Attributes说明：

model 表单数据对象

rules 表单验证规则

Form-Item Attributes说明：

prop 表单域 model 字段，在使用 validate、resetFields 方法的情况下，该属性是必填的

label 标签文本

详情属性及事件参考<http://element.eleme.io/#/zh-CN/component/form>

5、数据对象

```
data(){
  return {
    //站点列表
    siteList:[],
    //模版列表
    templateList:[],
    //新增界面数据
    pageForm: {
      siteId:'',
      templateId:'',
      pageName: '',
      pageAliase: '',
      pageWebPath: '',
      pageParameter:'',
      pagePhysicalPath:'',
      pageType:'',
      pageCreateTime: new Date()
    }
  }
},
methods:{
  addSubmit(){
    alert("提交")
  }
}
```

6、站点及模板数据（先使用静态数据）

在created钩子方法中定义，created是在html渲染之前执行，这里推荐使用created。

```
created:function(){
  //初始化站点列表
  this.siteList = [
    {
      siteId:'5a751fab6abb5044e0d19ea1',
      siteName:'门户主站'
    },
    {
      siteId:'102',
```


```
    siteName: '测试站'
  }
]
//模板列表
this.templateList = [
  {
    templateId: '5a962b52b00ffc514038faf7',
    templateName: '首页'
  },
  {
    templateId: '5a962bf8b00ffc514038fafa',
    templateName: '轮播图'
  }
]
}
```

7、测试预览

新增页面按钮：



页面列表

请选择站点  页面名称：

#	页面名称	别名	页面类型	访问路径
---	------	----	------	------

新增页面表单：

添加页面

所属站点

* 选择模版

* 页面名称

别名

* 访问路径

* 物理路径

类型 ☐ 静态 ☐ 动态

创建时间

2.3.1.2 添加返回

进入新增页面后只能通过菜单再次进入页面列表，可以在新增页面添加“返回”按钮，点击返回按钮返回到页面列表。

1) 新增页面按钮带上参数

```
<router-link class="mui-tab-item" :to="{path: '/cms/page/add/', query: {  
  page: this.params.page,  
  siteId: this.params.siteId}}">  
  <el-button type="primary" size="small">新增页面</el-button>  
</router-link>
```

说明：query表示在路由url上带上参数

2) 定义返回方法

在page_add.vue上定义返回按钮

```
<el-button type="primary" @click="go_back" >返回</el-button>
```

在page_add.vue上定义返回方法

```
go_back(){  
  this.$router.push({  
    path: '/cms/page/list', query: {  
      page: this.$route.query.page,  
      siteId: this.$route.query.siteId  
    }  
  })  
}
```

说明：this.\$route.query 表示取出路由上的参数列表，有两个取路由参数的方法：

- a、通过在路由上添加key/value串使用this.\$route.query来取参数，例如：/router1?id=123 ,/router1?id=456 可以通过this.\$route.query.id获取参数id的值。
- b、通过将参数作为路由一部分进行传参数使用this.\$route.params来获取，例如：定义的路由为/router1/:id ，请求/router1/123时可以通过this.\$route.params.id来获取，此种情况用this.\$route.query.id是拿不到的。

3) 查询列表支持回显

进入查询列表，从url中获取页码和站点id并赋值给数据模型对象，从而实现页面回显。

url例子：<http://localhost:12000/#/cms/page/list?page=2&siteId=5a751fab6abb5044e0d19ea1>

```
created() {  
  //从路由上获取参数  
  this.params.page = Number.parseInt(this.$route.query.page||1);  
  this.params.siteId = this.$route.query.siteId||'';  
  .....
```

小技巧：使用 || 返回第一个有效值

2.3.1.3 表单校验

1、配置校验规则：

Element-UI的Form组件提供表单校验的方法：

在form属性上配置rules（表单验证规则）

```
<el-form :model="pageForm" :rules="pageFormRules" label-width="80px" >
```

在数据模型中配置校验规则：

```
pageFormRules: {
  siteId: [
    {required: true, message: '请选择站点', trigger: 'blur'}
  ],
  templateId: [
    {required: true, message: '请选择模版', trigger: 'blur'}
  ],
  pageName: [
    {required: true, message: '请输入页面名称', trigger: 'blur'}
  ],
  pageWebPath: [
    {required: true, message: '请输入访问路径', trigger: 'blur'}
  ],
  pagePhysicalPath: [
    {required: true, message: '请输入物理路径', trigger: 'blur'}
  ]
}
```

更多的校验规则参考<http://element.eleme.io/#/zh-CN/component/form>中“表单验证”的例子。

2、点击提交按钮触发校验

1)在form表单上添加 ref属性（ref="pageForm"）在校验时引用此表单对象

```
<el-form :model="pageForm" :rules="pageFormRules" label-width="80px" ref="pageForm">
```

2) 执行校验

```
this.$refs.pageForm.validate((valid) => {  
  if (valid) {  
    alert('提交');  
  } else {  
    alert('校验失败');  
    return false;  
  }  
})
```

2.3.2 Api调用

1、在cms.js中定义page_add方法。

```
/*页面添加*/  
export const page_add = params => {  
  return http.requestPost(apiUrl+'/cms/page/add',params)  
}
```

2、添加事件

本功能使用到两个UI组件：

1、使用element-ui的message-box组件弹出确认提交窗口（<http://element.eleme.io/#/zh-CN/component/message-box>）。

```
this.$confirm('确认提交吗？', '提示', {}).then(() => { })
```

2、使用 message组件提示操作结果（<http://element.eleme.io/#/zh-CN/component/message>）

```
this.$message({  
  message: '提交成功',  
  type: 'success'  
})
```

完整的代码如下：

```
addSubmit(){  
  this.$refs.pageForm.validate((valid) => {  
    if (valid) {  
      this.$confirm('确认提交吗？', '提示', {}).then(() => {  
        cmsApi.page_add(this.pageForm).then((res) => {  
          console.log(res);  
          if(res.success){  
            this.$message({  
              message: '提交成功',
```

```
        type: 'success'
    });
    this.$refs['pageForm'].resetFields();
  }else{
    this.$message.error('提交失败');
  }
});
});
}
});
}
```

下边是测试：

1、进入页面列表页面

页面列表						
请选择站点		页面名称：		查询	新增页面	
#	页面名称	别名	页面类型	访问路径	物理路径	创建时间
1	index.html	首页	0	/index.html	F:\develop\xc_portal_static\	1517636273256
2	index_banner.html	轮播图	0	/include/index_banner.html	F:\develop\xc_portal_static\include\	1517902461255
1 2 3 4 5 6 ... 13						

2、点击“增加页面”按钮

添加页面

* 所属站点 请选择站点

* 选择模版 请选择

* 页面名称

别名

* 访问路径

* 物理路径

类型 ☐ 静态 ☐ 动态

创建时间 2018-06-11 13:24:00

提交 返回

3、输入页面信息点击“提交”

3 修改页面

修改页面用户操作流程：

- 1、用户进入修改页面，在页面上显示了修改页面的信息
- 2、用户修改页面的内容，点击“提交”，提示“修改成功”或“修改失败”

3.1 修改页面接口定义

修改页面需要定义的API如下：

```
@ApiOperation("通过ID查询页面")
public CmsPage findById(String id);

@ApiOperation("修改页面")
public CmsPageResult edit(String id,CmsPage cmsPage);
```

说明：提交数据使用post、put都可以，只是根据http方法的规范，put方法是对服务器指定资源进行修改，所以这里使用put方法对页面修改进行修改。

3.2 修改页面服务端开发

3.2.1Dao

使用 Spring Data提供的findById方法完成根据主键查询。

使用 Spring Data提供的save方法完成数据保存。

3.2.2Service

```
//根据id查询页面
public CmsPage getById(String id){
    Optional<CmsPage> optional = cmsPageRepository.findById(id);
    if(optional.isPresent()){
        return optional.get();
    }
    //返回空
    return null;
}

//更新页面信息
public CmsPageResult update(String id,CmsPage cmsPage) {
    //根据id查询页面信息
    CmsPage one = this.getById(id);
    if (one != null) {
        //更新模板id
```

```
one.setTemplateId(cmsPage.getTemplateId());
//更新所属站点
one.setSiteId(cmsPage.getSiteId());
//更新页面别名
one.setPageAliase(cmsPage.getPageAliase());
//更新页面名称
one.setPageName(cmsPage.getPageName());
//更新访问路径
one.setPageWebPath(cmsPage.getPageWebPath());
//更新物理路径
one.setPagePhysicalPath(cmsPage.getPagePhysicalPath());
//执行更新
CmsPage save = cmsPageRepository.save(one);
if (save != null) {
    //返回成功
    CmsPageResult cmsPageResult = new CmsPageResult(CommonCode.SUCCESS, save);
    return cmsPageResult;
}
//返回失败
return new CmsPageResult(CommonCode.FAIL, null);
}
```

3.2.3 Controller

1、根据id查询页面

```
@Override
@GetMapping("/get/{id}")
public CmsPage findById(@PathVariable("id") String id) {
    return pageService.getById(id);
}
```

2、保存页面信息

```
@Override
@PutMapping("/edit/{id}")//这里使用put方法，http方法中put表示更新
public CmsPageResult edit(@PathVariable("id") String id, @RequestBody CmsPage cmsPage) {
    return pageService.update(id, cmsPage);
}
```

3.3 修改页面前端开发

3.3.1 页面处理流程

页面的处理流程如下：

- 1、进入页面，通过钩子方法请求服务端获取页面信息，并赋值给数据模型对象
- 2、页面信息通过数据绑定在表单显示
- 3、用户修改信息点击“提交”请求服务端修改页面信息接口

3.3.3 修改页面

3.3.3.1 编写page_edit页面

修改页面的布局同添加页面，可以直接复制添加页面，在添加页面基础上修改。

下边编写页面内容：

- 1、编写page_edit.vue

页面布局同添加页面，略。

- 2、配置路由

进入修改页面传入pageId

```
import page_edit from '@module/cms/page/page_edit.vue';
{ path: '/cms/page/edit/:pageId', name: '修改页面', component: page_edit, hidden: true },
```

- 3、在页面列表添加“编辑”链接

参考table组件的例子，在page_list.vue上添加“操作”列

```
<el-table-column label="操作" width="80">
  <template slot-scope="page">
    <el-button
      size="small" type="text"
      @click="edit(page.row.pageId)">编辑
    </el-button>
  </template>
</el-table-column>
```

编写edit方法

```
//修改
edit:function (pageId) {
  this.$router.push({ path: '/cms/page/edit/'+pageId, query:{
    page: this.params.page,
    siteId: this.params.siteId}})
}
```

4、测试预览

页面列表

请选择站点	页面名称:	查询	新增页面				
#	页面名称	别名	页面类型	访问路径	物理路径	创建时间	操作
1	index.html	首页	0	/index.html	F:\develop\xc_portal_static\	1517636273256	编辑
2	index_banner.html	轮播图	0	/include/index_banner.html	F:\develop\xc_portal_static\include\	1517902461255	编辑

点击“编辑”打开修改页面窗口。

3.3.3.2 页面内容显示

本功能实现：进入修改页面立即显示要修改的页面信息。

1、定义api方法

```

/*页面查询*/
export const page_get = id => {
    return http.requestQuickGet(apiUrl+'/cms/page/get/'+id)
}
    
```

2、定义数据对象

进入修改页面传入pageId参数，在数据模型中添加pageId。

```

data(){
    return {
        .....
        //页面id
        pageId:'',
        .....
    }
}
    
```

3、在created钩子方法 中查询页面信息

```
created: function () {  
  //页面参数通过路由传入，这里通过this.$route.params来获取  
  this.pageId=this.$route.params.pageId;  
  //根据主键查询页面信息  
  cmsApi.page_get(this.pageId).then((res) => {  
    console.log(res);  
    if(res.success){  
      this.pageForm = res.cmsPage;  
    }  
  });  
}
```

7、预览页面回显效果

修改页面

* 所属站点

* 选择模版

* 页面名称

别名

* 访问路径

* 物理路径

类型 ☒ 静态 ☐ 动态

创建时间

3.3.4 Api调用

1、定义api方法

```
/*页面修改，采用put方法*/  
export const page_edit = (id,params) => {  
  return http.requestPut(apiUrl+'/cms/page/edit/'+id,params)  
}
```

2、提交按钮方法

添加提交按钮事件：

```
<el-button type="primary" @click="editSubmit">提交</el-button>
```

3、提交按钮事件内容：

```
editSubmit(){
  this.$refs.pageForm.validate((valid) => {
    if (valid) {
      this.$confirm('确认提交吗?', '提示', {}).then(() => {
        cmsApi.page_edit(this.pageId, this.pageForm).then((res) => {
          console.log(res);
          if (res.success) {
            this.$message({
              message: '修改成功',
              type: 'success'
            });
            //自动返回
            this.go_back();
          } else {
            this.$message.error('修改失败');
          }
        });
      });
    }
  });
}
```

4、测试

修改页面信息，点击提交。

4 删除页面

用户操作流程：

- 1、用户进入用户列表，点击“删除”
- 2、执行删除操作，提示“删除成功”或“删除失败”

4.1 删除页面接口定义

```
@ApiOperation("通过ID删除页面")
public ResponseResult delete(String id);
```

4.2 删除页面服务端开发

4.2.1 Dao

使用 Spring Data提供的deleteById方法完成删除操作。

4.2.2 Service

```
//删除页面
public ResponseResult delete(String id){
    CmsPage one = this.getById(id);
    if(one!=null){
        //删除页面
        cmsPageRepository.deleteById(id);
        return new ResponseResult(CommonCode.SUCCESS);
    }
    return new ResponseResult(CommonCode.FAIL);
}
```

4.2.3 Controller

```
@DeleteMapping("/del/{id}") //使用http的delete方法完成岗位操作
public ResponseResult delete(@PathVariable("id") String id) {
    return pageService.delete(id);
}
```

4.3 删除页面前端开发

4.3.1 Api方法

```
/*页面删除*/
export const page_del = id => {
    return http.requestDelete(apiUrl+'cms/page/del/'+id)
}
```

4.3.2 编写页面

1、在page_list.vue页面添加删除按钮

```
<el-table-column label="操作" width="120">
  <template slot-scope="page">
    <el-button
      size="small" type="text"
      @click="edit(page.row.pageId)">编辑
    </el-button>
    <el-button
      size="small" type="text"
      @click="del(page.row.pageId)">删除
    </el-button>
  </template>
</el-table-column>
```

2、删除事件

```
//删除
del:function (pageId) {
  this.$confirm('确认删除此页面吗?', '提示', {}).then(() => {
    cmsApi.page_del(pageId).then((res)=>{
      if(res.success){
        this.$message({
          type: 'success',
          message: '删除成功!'
        });
        //查询页面
        this.query()
      }else{
        this.$message({
          type: 'error',
          message: '删除失败!'
        });
      }
    })
  })
}
```

5 异常处理

5.1 异常处理的问题分析

从添加页面的service方法中找问题：

```
//添加页面
public CmsPageResult add(CmsPage cmsPage){
  //校验页面是否存在，根据页面名称、站点Id、页面webpath查询

  CmsPage cmsPage1 =
```




```
cmsPageRepository.findByPageNameAndSiteIdAndPageWebPath(cmsPage.getPageName(),
cmsPage.getSiteId(), cmsPage.getPageWebPath());
    if(cmsPage1==null){
        cmsPage.setPageId(null);//添加页面主键由spring data 自动生成
        cmsPageRepository.save(cmsPage);
        //返回结果
        CmsPageResult cmsPageResult = new CmsPageResult(CommonCode.SUCCESS,cmsPage);
        return cmsPageResult;
    }
    return new CmsPageResult(CommonCode.FAIL,null);
}
```

问题：

- 1、上边的代码只要操作不成功仅向用户返回“错误代码：11111，失败信息：操作失败”，无法区别具体的错误信息。
- 2、service方法在执行过程出现异常在哪捕获？在service中需要都加try/catch，如果在controller也需要添加try/catch，代码冗余严重且不易维护。

解决方案：

- 1、在Service方法中的编码顺序是先校验判断，有问题则抛出具体的异常信息，最后执行具体的业务操作，返回成功信息。
- 2、在统一异常处理类中去捕获异常，无需controller捕获异常，向用户返回统一规范的响应信息。

代码模板如下：

```
//添加页面
public CmsPageResult add(CmsPage cmsPage){
    //校验cmsPage是否为空
    if(cmsPage == null){
        //抛出异常，非法请求
        //...
    }
    //根据页面名称查询（页面名称已在mongodb创建了唯一索引）
    CmsPage cmsPage1 =
cmsPageRepository.findByPageNameAndSiteIdAndPageWebPath(cmsPage.getPageName(),
cmsPage.getSiteId(), cmsPage.getPageWebPath());
    //校验页面是否存在，已存在则抛出异常
    if(cmsPage1 !=null){
        //抛出异常，已存在相同的页面名称
        //...
    }
    cmsPage.setPageId(null);//添加页面主键由spring data 自动生成
    CmsPage save = cmsPageRepository.save(cmsPage);
    //返回结果
    CmsPageResult cmsPageResult = new CmsPageResult(CommonCode.SUCCESS,save);
    return cmsPageResult;
}
```

5.2 异常处理流程

系统对异常的处理使用统一的异常处理流程：

- 1、自定义异常类型。
- 2、自定义错误代码及错误信息。
- 3、对于可预知的异常由程序员在代码中主动抛出，由SpringMVC统一捕获。

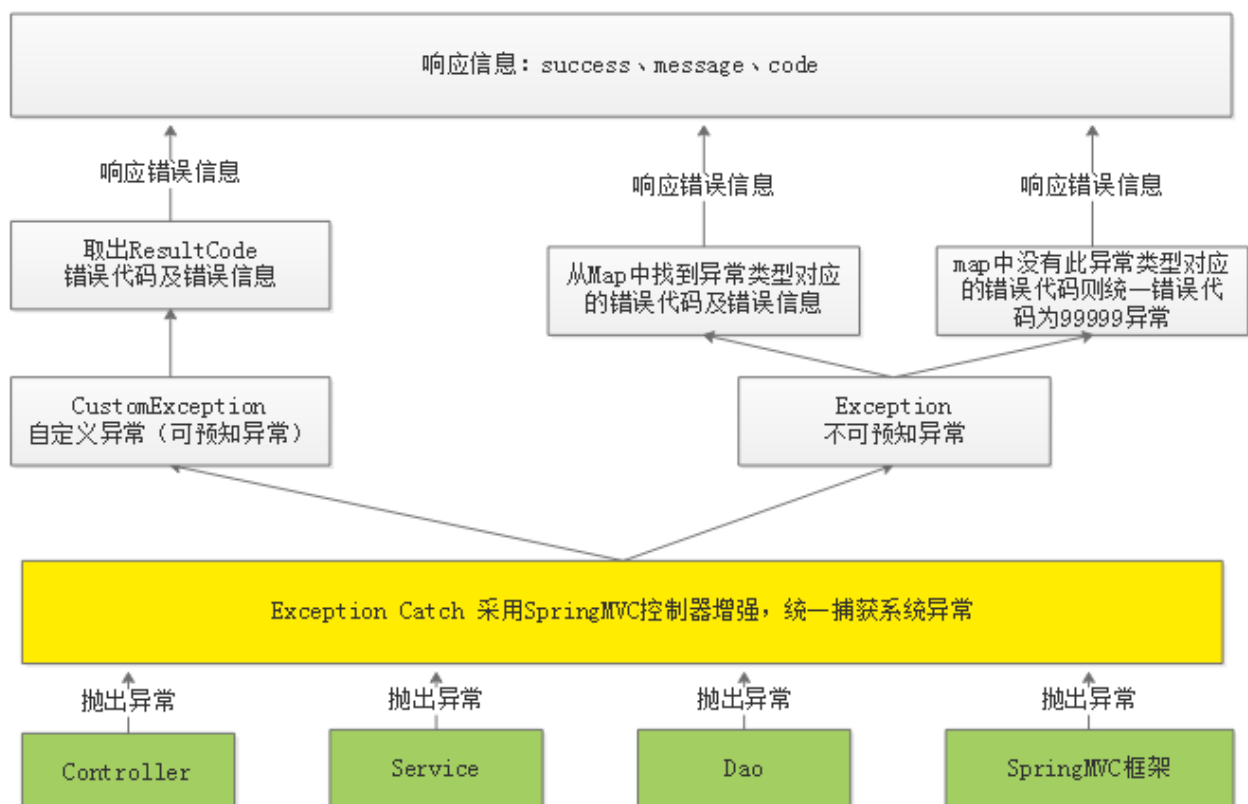
可预知异常是程序员在代码中手动抛出本系统定义的特定异常类型，由于是程序员抛出的异常，通常异常信息比较齐全，程序员在抛出时会指定错误代码及错误信息，获取异常信息也比较方便。

- 4、对于不可预知的异常（运行时异常）由SpringMVC统一捕获Exception类型的异常。

不可预知异常通常是由于系统出现bug、或一些不要抗拒的错误（比如网络中断、服务器宕机等），异常类型为RuntimeException类型（运行时异常）。

- 5、可预知的异常及不可预知的运行时异常最终会采用统一的信息格式（错误代码+错误信息）来表示，最终也会随请求响应给客户端。

异常抛出及处理流程：



- 1、在controller、service、dao中程序员抛出自定义异常；springMVC框架抛出框架异常类型
- 2、统一由异常捕获类捕获异常，并进行处理
- 3、捕获到自定义异常则直接取出错误代码及错误信息，响应给用户。

- 4、捕获到非自定义异常类型首先从Map中找该异常类型是否对应具体的错误代码，如果有则取出错误代码和错误信息并响应给用户，如果从Map中找不到异常类型所对应的错误代码则统一为99999错误代码并响应给用户。
- 5、将错误代码及错误信息以Json格式响应给用户。

5.3 可预知异常处理

5.3.1 自定义异常类

在common工程定义异常类型。

```
package com.xuecheng.framework.exception;

import com.xuecheng.framework.model.response.ResultCode;

public class CustomException extends RuntimeException {

    private ResultCode resultCode;

    public CustomException(ResultCode resultCode) {
        //异常信息为错误代码+异常信息
        super("错误代码：" + resultCode.code() + "错误信息：" + resultCode.message());
        this.resultCode = resultCode;
    }

    public ResultCode getResultCode(){
        return this.resultCode;
    }
}
```

5.3.2 异常抛出类

```
package com.xuecheng.framework.exception;

import com.xuecheng.framework.model.response.ResultCode;

public class ExceptionCast {

    //使用此静态方法抛出自定义异常
    public static void cast(ResultCode resultCode){
        throw new CustomException(resultCode);
    }
}
```

5.3.3 异常捕获类

使用 @ControllerAdvice和@ExceptionHandler注解来捕获指定类型的异常

```
package com.xuecheng.framework.exception;

import com.xuecheng.framework.model.response.ResponseResult;
import com.xuecheng.framework.model.response.ResultCode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

@ControllerAdvice
public class ExceptionCatch {
    private static final Logger LOGGER = LoggerFactory.getLogger(ExceptionCatch.class);

    //捕获 CustomException异常
    @ExceptionHandler(CustomException.class)
    @ResponseBody
    public ResponseResult customException(CustomException e) {
        LOGGER.error("catch exception : {}\\r\\nexception: ", e.getMessage(), e);
        ResultCode resultCode = e.getResultCode();
        ResponseResult responseResult = new ResponseResult(resultCode);

        return responseResult;
    }
}
```

5.3.4 异常处理测试

5.3.4.1 定义错误代码

每个业务操作的异常使用异常代码去标识。

```
package com.xuecheng.framework.domain.cms.response;

import com.xuecheng.framework.model.response.ResultCode;
import lombok.ToString;

@ToString
public enum CmsCode implements ResultCode {
    CMS_ADDPAGE_EXISTS(false, 24001, "页面已存在!");
    //操作结果
}
```

```
boolean success;
//操作代码
int code;
//提示信息
String message;
private CmsCode(boolean success, int code, String message){
    this.success = success;
    this.code = code;
    this.message = message;
}

@Override
public boolean success() {
    return success;
}

@Override
public int code() {
    return code;
}

@Override
public String message() {
    return message;
}
}
```

5.3.4.2 异常处理测试

1、抛出异常

在controller、service、dao中都可以抛出异常。

修改PageService的add方法，添加抛出异常的代码

```
        //校验页面是否存在，根据页面名称、站点Id、页面webpath查询
        CmsPage cmsPage1 =
cmsPageRepository.findByPageNameAndSiteIdAndPageWebPath(cmsPage.getPageName(),
cmsPage.getSiteId(), cmsPage.getPageWebPath());
        if(cmsPage1 !=null){
            //校验页面是否存在，已存在则抛出异常
            ExceptionCast.cast(CmsCode.CMS_ADDPAGE_EXISTS);
        }
```

2、启动工程，扫描到异常捕获的类ExceptionCatch

在springBoot的启动类中添加

```
@ComponentScan(basePackages="com.xuecheng.framework")//扫描common工程下的类
```

3、前端展示异常信息

服务端响应信息如下：

```
▼ {success: false, code: 24001, message: "页面名称已存在!"} ⓘ  
  code: 24001  
  message: "页面名称已存在!"  
  success: false  
  __proto__: Object
```

页面提取异常处理

```
addSubmit(){  
  this.$refs.pageForm.validate((valid) => {  
    if (valid) {  
      this.$confirm('确认提交吗?', '提示', {}).then(() => {  
        cmsApi.page_add(this.pageForm).then((res) => {  
          console.log(res);  
          if(res.success){  
            this.$message({  
              message: '提交成功',  
              type: 'success'  
            });  
            this.$refs['pageForm'].resetFields();  
          }else if(res.message){  
            this.$message.error(res.message);  
          }else{  
            this.$message.error('提交失败');  
          }  
        });  
      });  
    }  
  });  
}
```

5.4 不可预知异常处理

5.4.1 定义异常捕获方法

5.4.1.1 异常抛出测试

使用postman测试添加页面，不输入cmsPost信息，提交，报错信息如下：

```
org.springframework.http.converter.HttpMessageNotReadableException  
此异常是springMVC在进行参数转换时报的错误。
```

具体的响应的信息为：

```
{
  "timestamp": 1528712906727,
  "status": 400,
  "error": "Bad Request",
  "exception": "org.springframework.http.converter.HttpMessageNotReadableException",
  "message": "Required request body is missing: public
com.xuecheng.framework.domain.cms.response.CmsPageResult
com.xuecheng.manage_cms.web.controller.CmsPageController.add(com.xuecheng.framework.domain.cms.C
msPage)",
  "path": "/cms/page/add"
}
```

上边的响应信息在客户端是无法解析的。

在异常捕获类中添加对Exception异常的捕获：

```
@ExceptionHandler(Exception.class)
@ResponseBody
public ResponseResult exception(Exception exception){
    //记录日志
    LOGGER.error("catch exception:{}",exception.getMessage());

    return null;
}
```

5.4.1.2 异常捕获方法

针对上边的问题其解决方案是：

- 1、我们在map中配置HttpMessageNotReadableException和错误代码。
- 2、在异常捕获类中对Exception异常进行捕获，并从map中获取异常类型对应的错误代码，如果存在错误代码则返回此错误，否则统一返回99999错误。

具体的开发实现如下：

- 1、在通用错误代码类CommCode中配置非法参数异常

```
INVALID_PARAM(false,10003,"非法参数！"),
```

- 2、在异常捕获类中配置 HttpMessageNotReadableException 为非法参数异常。

异常捕获类代码如下：

```
package com.xuecheng.framework.exception;

import com.google.common.collect.ImmutableMap;
```



```
import com.xuecheng.framework.model.response.CommonCode;
import com.xuecheng.framework.model.response.ResponseResult;
import com.xuecheng.framework.model.response.ResultCode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.converter.HttpMessageNotReadableException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * @author Administrator
 * @version 1.0
 * @create 2018-06-11 17:16
 */
@ControllerAdvice
public class ExceptionCatch {
    private static final Logger LOGGER = LoggerFactory.getLogger(ExceptionCatch.class);

    //使用EXCEPTIONS存放异常类型和错误代码的映射，ImmutableMap的特点一旦创建不可改变，并且线程安全
    private static ImmutableMap<Class<? extends Throwable>, ResultCode> EXCEPTIONS;
    //使用builder来构建一个异常类型和错误代码的异常
    protected static ImmutableMap.Builder<Class<? extends Throwable>, ResultCode> builder =
    ImmutableMap.builder();

    //捕获Exception异常
    @ResponseBody
    @ExceptionHandler(Exception.class)
    public ResponseResult exception(Exception e) {
        LOGGER.error("catch exception : {}\\r\\nexception: ", e.getMessage(), e);
        if(EXCEPTIONS == null)
            EXCEPTIONS = builder.build();
        final ResultCode resultCode = EXCEPTIONS.get(e.getClass());
        final ResponseResult responseResult;
        if (resultCode != null) {
            responseResult = new ResponseResult(resultCode);
        } else {
            responseResult = new ResponseResult(CommonCode.SERVER_ERROR);
        }
        return responseResult;
    }

    //捕获 CustomException异常
    @ExceptionHandler(CustomException.class)
    @ResponseBody
    public ResponseResult customException(CustomException e) {
        LOGGER.error("catch exception : {}\\r\\nexception: ", e.getMessage(), e);
        ResultCode resultCode = e.getResultCode();
        ResponseResult responseResult = new ResponseResult(resultCode);

        return responseResult;
    }
}
```



```
static{
    //在这里加入一些基础的异常类型判断
    builder.put(HttpMessageNotReadableException.class,CommonCode.INVALIDPARAM);
}
}
```

5.4.3 异常处理测试

仍然模拟“问题测试”中的测试步骤，异常结果为“非法参数”。

6 实战

此部分为自学内容，根据今天所学知识完成下边的任务。

6.1 查询条件完善

页面查询条件增加：页面名称、页面类型。

页面名称对应CmsPage模型类中的pageName属性。

页面类型对应CmsPage模型类中的pageType属性。

查询要求：

页面名称：模糊查询

页面类型：精确匹配，页面类型包括：静态和动态，在数据库中静态用“0”表示，动态用“1”表示。

6.2 页面属性增加DataUrl

在CmsPage.java模型类型中有一个dataUrl属性，此属性在页面静态化时需要填写。

本需求要求：

- 1、在新增页面增加dataUrl输入框，并支持添加。
- 2、在修改页面增加dataUrl输入框，并支持修改。