

Day18

添加项目所需的静态资源

预约管理模块 - 检查项管理

检查项 - 新增功能

修改前台页面

弹出新增窗口

输入校验

提交表单数据

添加后台代码

控制层

服务接口

服务实现类

Dao接口

Mapper映射文件

检查项 - 分页功能

修改前台页面

定义分页相关模型数据

定义分页方法

分页查询

完善分页方法执行时机

添加后台代码

控制层

服务层接口

服务层实现类

Dao接口

Mapper映射文件

检查项 - 删除功能

检查项 - 编辑功能

添加项目所需的静态资源

当前，我们后台界面的搭建需要使用到 html、js、css、图片等静态资源，所有这些资源放到 seehope-health-backend 工程中。

预约管理模块 – 检查项管理

预约管理模块包括如下功能：

- 检查项管理
- 检查组管理
- 体检套餐管理
- 预约设置等。

总的来说，预约管理属于系统的基础功能，主要就是管理一些体检的基础数据。

检查项 – 新增功能

修改前台页面

检查项管理页面对应的是 checkitem.html 页面，根据产品设计的原型已经完成了页面基本结构的编写，现在根据需求完善其页面动态效果。

弹出新增窗口

页面中已经提供了新增窗口，只是处于隐藏状态。只需要将控制展示状态的属性 dialogFormVisible 改为 true 就可以显示出新增窗口。

新建按钮绑定的方法为 handleCreate，所以在 handleCreate 方法中修改 dialogFormVisible 属性的值为 true 即可。同时为了增加用户体验度，需要每次点击新建按钮时清空表单输入项。

JavaScript |  复制代码

```
1  // 重置表单
2  resetForm() {
3      this.formData = {};
4  },
5  // 弹出添加窗口
6  handleCreate() {
7      this.resetForm();
8      this.dialogFormVisible = true;
9  }
```

输入校验

JavaScript | 复制代码

```
1 // 校验规则
2 rules: {
3   code: [{ required: true, message: '项目编码为必填项', trigger: 'blur' }],
4   name: [{ required: true, message: '项目名称为必填项', trigger: 'blur' }]
5 }
```

提交表单数据

点击新增窗口中的确定按钮时，触发handleAdd方法，所以需要在handleAdd方法中进行完善。

Vue | 复制代码

```
1 // 添加
2 handleAdd() {
3   // 进行表单校验
4   this.$refs['dataAddForm'].validate((valid) => {
5     if (valid) {
6       // 表单校验通过，发生ajax请求，将录入的数据提交到后台进行处理
7       console.log(this.formData);
8       axios.post("/checkitem/add.do", this.formData).then((res) => {
9         // 关闭新增窗口
10        this.dialogFormVisible = false;
11        // 执行成功
12        if (res.data.flag) {
13          // 新增成功后，重新调用分页查询方法，查询出最新的数据
14          this.findPage();
15          // 弹出提示信息
16          this.$message({
17            message: res.data.message,
18            type: 'success'
19          });
20          // 执行失败
21        } else {
22          // 弹出提示
23          this.$message.error(res.data.message);
24        }
25      });
26      // 校验不通过
27    } else {
28      this.$message.error("数据校验失败，请检查你的输入信息是否正确！");
29      return false;
30    }
31  });
32 },
```

添加后台代码

控制层

在 seehope-health-backend 工程中创建 CheckItemController 类。

Java  复制代码

```
1  package com.rushuni.controller;
2
3  import com.rushuni.constant.MessageConstant;
4  import com.rushuni.entity.PageResult;
5  import com.rushuni.entity.QueryPageBean;
6  import com.rushuni.entity.Result;
7  import com.rushuni.pojo.CheckItem;
8  import com.rushuni.service.CheckItemService;
9  import org.apache.dubbo.config.annotation.DubboReference;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 /**
15  * @author rushuni
16  * @date 2021/08/03
17  */
18 @RestController
19 @RequestMapping("/checkitem")
20 public class CheckItemController {
21     @DubboReference//查找服务
22     private CheckItemService checkItemService;
23
24     //新增检查项
25     @RequestMapping("/add")
26     public Result add(@RequestBody CheckItem checkItem){
27         try{
28             checkItemService.add(checkItem);
29         }catch (Exception e){
30             e.printStackTrace();
31             //服务调用失败
32             return new Result(false, MessageConstant.ADD_CHECKITEM_FAIL);
33         }
34         return new Result(true, MessageConstant.ADD_CHECKITEM_SUCCESS);
35     }
36 }
37
```

服务接口

在 seehope-health-interface 工程中创建 CheckItemService 接口

```
1 package com.rushuni.service;
2
3 import com.rushuni.pojo.CheckItem;
4
5 /**
6  * @author rushuni
7  * @date 2021/08/03
8  */
9 public interface CheckItemService {
10     /**
11      * 添加检查项
12      * @param checkItem
13      */
14     void add(CheckItem checkItem);
15 }
```

服务实现类

在seehope-health-service-provider工程中创建CheckItemServiceImpl实现类

```
1 package com.rushuni.service.impl;
2
3 import com.rushuni.dao.CheckItemDao;
4 import com.rushuni.service.CheckItemService;
5 import org.apache.dubbo.config.annotation.DubboService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.transaction.annotation.Transactional;
8 import com.rushuni.pojo.CheckItem;
9
10 /**
11  * 检查项服务
12  * @author rushuni
13  * @date 2021/8/3
14  */
15 @DubboService(interfaceClass = CheckItemService.class)
16 @Transactional
17 public class CheckItemServiceImpl implements CheckItemService {
18     /**
19      * 注入DAO对象
20      */
21     @Autowired
22     private CheckItemDao checkItemDao;
23
24     /**
25      * 添加检查项
26      * @param checkItem
27      */
28     @Override
29     public void add(CheckItem checkItem) {
30         checkItemDao.add(checkItem);
31     }
32 }
```

Dao接口

在health_service_provider工程中创建CheckItemDao接口，本项目是基于Mybatis的Mapper代理技术实现持久层操作，故只需要提供接口和Mapper映射文件，无须提供实现类

```

1 package com.rushuni.dao;
2
3 import com.rushuni.pojo.CheckItem;
4
5 /**
6  * @author rushuni
7  * @date 2021/08/03
8  */
9 public interface CheckItemDao {
10     /**
11      * 添加检查项
12      * @param checkItem
13      */
14     void add(CheckItem checkItem);
15 }

```

Mapper映射文件

在health_service_provider工程中创建CheckItemDao.xml映射文件，需要和CheckItemDao接口在同一目录下

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3     "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
4 <mapper namespace="com.rushuni.dao.CheckItemDao">
5     <!--插入检查项数据-->
6     <insert id="add" parameterType="com.rushuni.pojo.CheckItem">
7         insert into
8         t_checkitem(code,name,sex,age,price,type,remark,attention)
9         values
10         (#{code},#{name},#{sex},#{age},#{price},#{type},#{remark},#
11         {attention})
12     </insert>
13 </mapper>

```

检查项 – 分页功能

本项目所有分页功能都是基于 ajax 的异步请求来完成的，请求参数和后台响应数据格式都使用json数据格式。

请求参数包括页码、每页显示记录数、查询条件。

请求参数的json格式为：{currentPage:1,pageSize:10,queryString:"itcast"}

后台响应数据包括总记录数、当前页需要展示的数据集合。

响应数据的json格式为：{total:1000,rows:[]}

修改前台页面

定义分页相关模型数据

```
1  // 分页相关模型数据
2  pagination: {
3    // 当前页码
4    currentPage: 1,
5    // 每页显示的记录数
6    pageSize: 10,
7    // 总记录数
8    total: 0,
9    // 查询条件
10   queryString: null
11 },
12 // 当前页要展示的分页列表数据
13 dataList: [],
```

Vue | 复制代码

定义分页方法

在页面中提供了 findPage 方法用于分页查询，为了能够在 checkitem.html 页面加载后直接可以展示分页数据，可以在 VUE 提供的钩子函数 created 中调用 findPage 方法

```
1  // 钩子函数，VUE对象初始化完成后自动执行
2  created() {
3    // VUE对象初始化完成后调用分页查询方法，完成分页查询
4    this.findPage();
5  },
```

Vue | 复制代码

分页查询


```

1  // 分页查询
2  findPage() {
3    // 发送ajax请求, 提交分页相关请求参数 (页码、每页显示记录数、查询条件)
4    var param = {
5      // 页码
6      currentPage: this.pagination.currentPage,
7      // 每页显示的记录数
8      pageSize: this.pagination.pageSize,
9      // 查询条件
10     queryString: this.pagination.queryString
11   };
12   axios.post("/checkitem/findPage.do", param).then((res) => {
13     // 解析Controller响应回的数据, 为模型数据赋值
14     this.pagination.total = res.data.total;
15     this.dataList = res.data.rows;
16   });
17 },

```

完善分页方法执行时机

除了在 created 钩子函数中调用 findPage 方法查询分页数据之外, 当用户点击查询按钮或者点击分页条中的页码时也需要调用 findPage 方法重新发起查询请求。

为查询按钮绑定单击事件, 调用 findPage 方法

```

1  <el-button @click="findPage()" class="dalfBut">查询</el-button>

```

为分页条组件绑定current-change事件, 此事件是分页条组件自己定义的事件, 当页码改变时触发, 对应的处理函数为 handleCurrentChange

```

1  <el-pagination class="pagiantion"
2    @current-change="handleCurrentChange"
3    :current-page="pagination.currentPage"
4    :page-size="pagination.pageSize"
5    layout="total, prev, pager, next, jumper"
6    :total="pagination.total">
7  </el-pagination>

```

定义 handleCurrentChange 方法

```
1 // 切换页码
2 handleCurrentChange(currentPage) {
3     // 设置最新的页码
4     this.pagination.currentPage = currentPage;
5     // 重新调用findPage方法进行分页查询
6     this.findPage();
7 },
```

添加后台代码

控制层

在 CheckItemController 中增加分页查询方法

```
1 //分页查询
2 @RequestMapping("/findPage")
3 public PageResult findPage(@RequestBody QueryPageBean queryPageBean){
4     PageResult pageResult = checkItemService.pageQuery(
5         queryPageBean.getCurrentPage(),
6         queryPageBean.getPageSize(),
7         queryPageBean.getQueryString());
8     return pageResult;
9 }
```

服务层接口

在CheckItemService服务接口中扩展分页查询方法

```
1 public PageResult pageQuery(Integer currentPage, Integer pageSize, String
    queryString);
```

服务层实现类

在 CheckItemServiceImpl 服务实现类中实现分页查询方法，基于 Mybatis分页助手插件实现分页


```
1 public PageResult pageQuery(Integer currentPage, Integer pageSize, String
    queryString) {
2     PageHelper.startPage(currentPage, pageSize);
3     Page<CheckItem> page = checkItemDao.selectByCondition(queryString);
4     return new PageResult(page.getTotal(), page.getResult());
5 }
```

Dao接口

在 CheckItemDao 接口中扩展分页查询方法

```
1 public Page<CheckItem> selectByCondition(String queryString);
```

Java


 复制代码

Mapper映射文件

在 CheckItemDao.xml 文件中增加SQL定义

```
1 <select id="selectByCondition" parameterType="string"
2         resultType="com.itheima.pojo.CheckItem">
3     select * from t_checkitem
4     <if test="value != null and value.length > 0">
5         where code = #{value} or name = #{value}
6     </if>
7 </select>
```

XML

 复制代码

检查项 – 删除功能

检查项 – 编辑功能