Bootstrap 开发框架
系统功能介绍白皮书

版本： 5.0

编制人：伍华聪

序号	修改人	修改日期	修改后版本	修改说明
1	伍华聪	2015-09-03	V1.0	正式版
2	伍华聪	2016-09-08	V2.0	修订版
3	伍华聪	2017-07-23	V3.0	修订版
4	伍华聪	2017-9-27	V4.0	更新列表排序等功能
5	伍华聪	2021-5-30	V5.0	增加 Vue&Element 内容介绍

目　录

1. 引言

1.1. 背景

这几年，我一直专注于开发框架的研究和应用，开发出来的产品，先后有《Winform 开发框架》、《WCF 开发框架》、《混合式开发框架》、《基于 EasyUI 的 Web 开发框架》、《微信开发框架》等框架以及一些项目的开发工作，这些框架技术在各自的领域发挥了极大的作用，特别是框架的案例应用结合代码生成工具的快速开发，给客户提供一个统一、可重用、快速高效的标准开发模式，为用户开发各种应用提供了快捷、稳定的平台和解决思路，同时也为各个企业培养了很多技术人员，帮助企业积累了很多基础应用开发的知识和经验。

这几年，随着 JQuery 技术的兴起，很多技术应用如雨后春笋般冒出来，因此我的《基于 EasyUI 的 Web 框架》也随之产生，这个框架很好利用了 Ajax 和 Jquery 及相关控件等技术，结合微软的 MVC 后台处理技术，在开发管理后台界面实现了非常不错的展示效果。而通过和代码生成工具 Database2Sharp 的整合，可以快速开发后台代码以及基于 MVC 的控制器和前端视图代码，因此可以极大提高开发效率，以及统一系统模块的界面样式。

随着 Web 前端技术的发展，Bootstrap 响应式技术框架也随之出现，这种响应式的技术框架，除了能够很好适应各种设备展示外，还提供了很多 CSS 的酷炫样式，在很多大企业以及前端应用里面，采用 Bootstrap 技术的网站和平台也越来越多，基于 Bootstrap 技术开发出来的插件也琳琅满目，因此可以结合开发出功能很强大的系统应用。Bootstrap 响应式技术框架除了可以制作界面美观、功能强大的后台管理界面外，还可以做适应在小设备上的微网站、购物商城等热门的 Web 应用。

基于技术研究和革新的目的，我们引入更新、更强大的 Bootstrap 和 Metronic 框架技术，整合之前的《基于 EasyUI 的 Web 开发框架》，继续使用成熟的 **MVC + Enterprise Library** 等相关的技术对原有的 Web 开发框架进行了升级。

另外我们在《Bootstrap 开发框架》高级版本里面，在常规的直接连接数据库的 Web 项目之外，增加了工作流的管理界面，实现工作流表单的审批、会签等多种流程处理过程，并结合代码生成工具快速生成相关的工作流界面。

技术的脚步永不停歇，随着 Web 端的纯前端技术的广泛使用，Vue 的语言逐渐在国内获得主导地位，Vue & Element 的搭配能够迅速开发后台管理系统，因此我们引入了 WebAPI + Vue&Element 的前端技术框架，基于同一个数据库的基础上，增加了 Vue & Element 的前端，相当于在原先 Bootstrap 的 Asp.net 的 BS 前端基础上增加多了一个管理后台的前端，以适应技术场景的选择，同时也丰富了框架的灵活性和功能性。

该《Bootstrap 开发框架》是我们经过多年的项目积累，吸收众多框架产品的前沿思路，以及客户的宝贵意见，反复提炼及优化而成的。

1.2. 编写目的

本文档主要介绍《基于 Metronic 的 Bootstrap 开发框架》以及《Vue & Element 后台管理前端》中系统的各种功能特性，介绍该框架的各种界面效果和重点功能的分析，以便阅读者能够更全面的、更真实了解《Bootstrap 开发框架》和《Vue & Element 后台管理前端框架》（以下统称框架）的相关特点。

1.3. 参考资料

序号	名称	版本/日期	来源
1	《Web 开发框架-架构设计说明书.doc》		内部
2	《Winform 开发框架-架构设计说明书.doc》		内部
3			内部

1.4. 术语缩写及约定

1 在本文件中出现的“Bootstrap开发框架”或“Web开发框架”一词，除非特别说明，均指基于**MVC + Metronic+Bootstrap + Enterprise Library**等相关的技术研制的Web开发框架。

2 在本文安装.NET框架中，除非特别说明，均指 **.NET 4.5** 框架。

3 本文的开发环境为Visual Studio 2017/2019 + VS Code，基于MVC5和Bootstrap3.x 的Web开发技术，以及基于Vue和Element的前端开发技术。

4 Bootstrap是一个前端的技术框架，很多平台都可以采用，JAVA/PHP/.NET都可以用

来做前端界面，整合JQuery可以实现非常丰富的界面效果，目前也有很多Bootstrap的插件能够提供给大家使用。

5 Metronic是一个国外的基于HTML、JS等技术的Bootstrap开发框架整合，整合了很多Bootstrap的前端技术和插件的使用，是一个非常不错的技术框架。

6 Vue & Element后台管理前端框架是一个纯前端的后台管理框架，集合Web API可以实现各种业务数据的展示和处理，该前端基于VS Code开发工具开发。

7 框架中的Bootstrap开发框架和Vue & Element前端框架都公用一个数据库，后端数据库支持多种不同的数据库类型，如MS SQLServer、Oracle、Mysql、SQLite、PostgreSQL等常规数据库。

2. Bootstrap 开发框架的重要特性总结

2.1. 框架总体介绍

1）技术特点

整个基于 Metronic 的 Bootstrap 开发框架，界面部分采用较新的 Bootstrap 技术，采用当前最新的 Bootstrap3.x，集成了众多功能强大的 Bootstrap 控件。

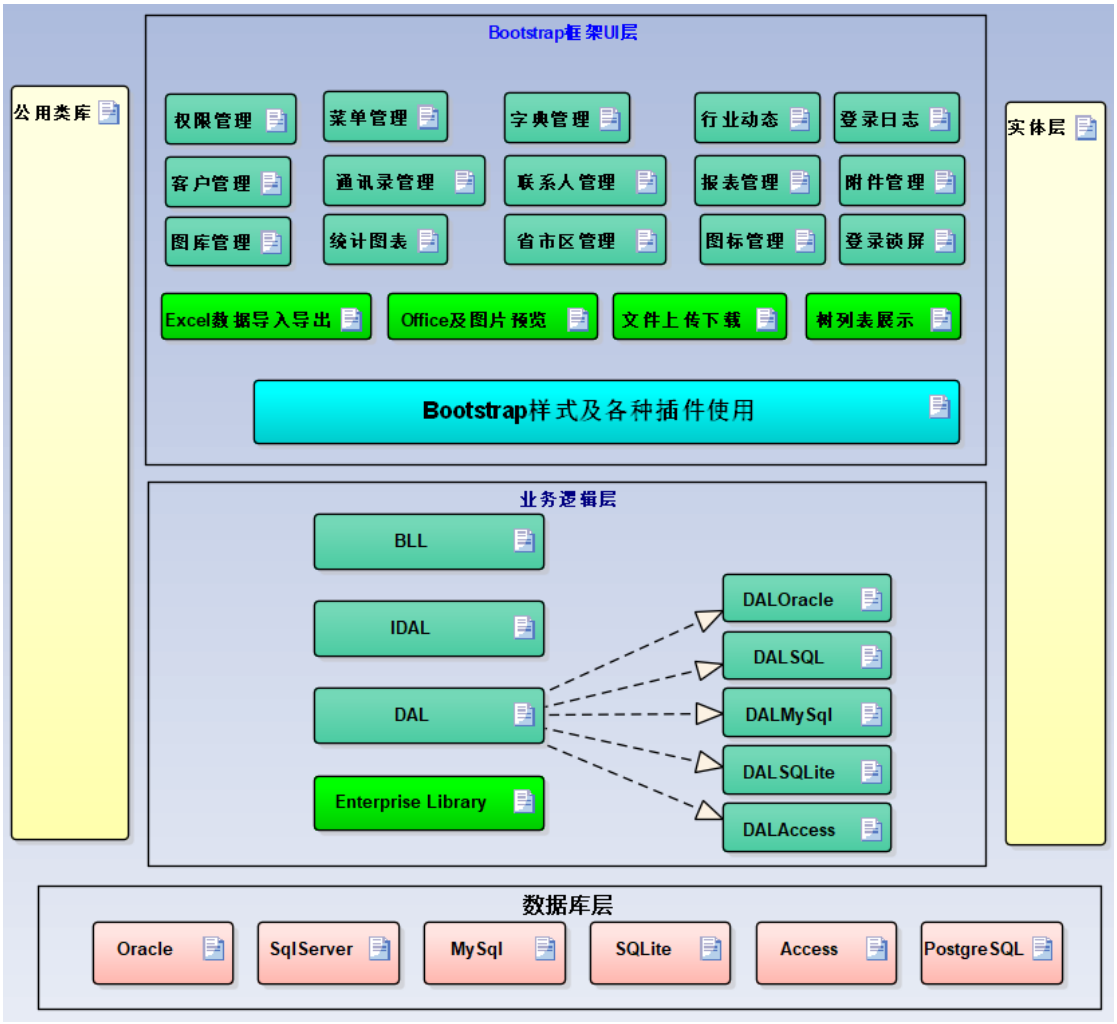
Bootstrap 是一个前端的技术框架，很多平台都可以采用，JAVA/PHP/.NET 都可以用来做前端界面，整合 JQuery 可以实现非常丰富的界面效果，目前也有很多 Bootstrap 的插件能够提供给大家使用，本框架集合了众多最为优秀的插件，能给我们 Web 的用户体验提升到一个前所未有的水平。

Metronic 是一个国外的基于 HTML、JS 等技术的 Bootstrap 开发框架整合，整合了很多 Bootstrap 的前端技术和插件的使用，是一个非常不错的技术框架。本框架以这个为基础，结合我对 MVC 的 Web 框架的研究，整合了基于 MVC 的 Bootstrap 开发框架，使之能够符合实际项目的结构需要。

框架后台采用基于 C#的 MVC 技术，是目前.NET 开发最为成熟流行的技术，框架后台数据库支持 Oracle、SqlServer、MySql、Sqlite、PostgreSQL、Access 等常规数据库，可通过配置进行自由切换，使用 Enterprise Library 模块进行数据访问的控制，使得数据访问更方便轻松。数据库可以同时使用不同类型的数据库，也可以把一个数据库切分为多个业务数据库，底层支持更加弹性化。

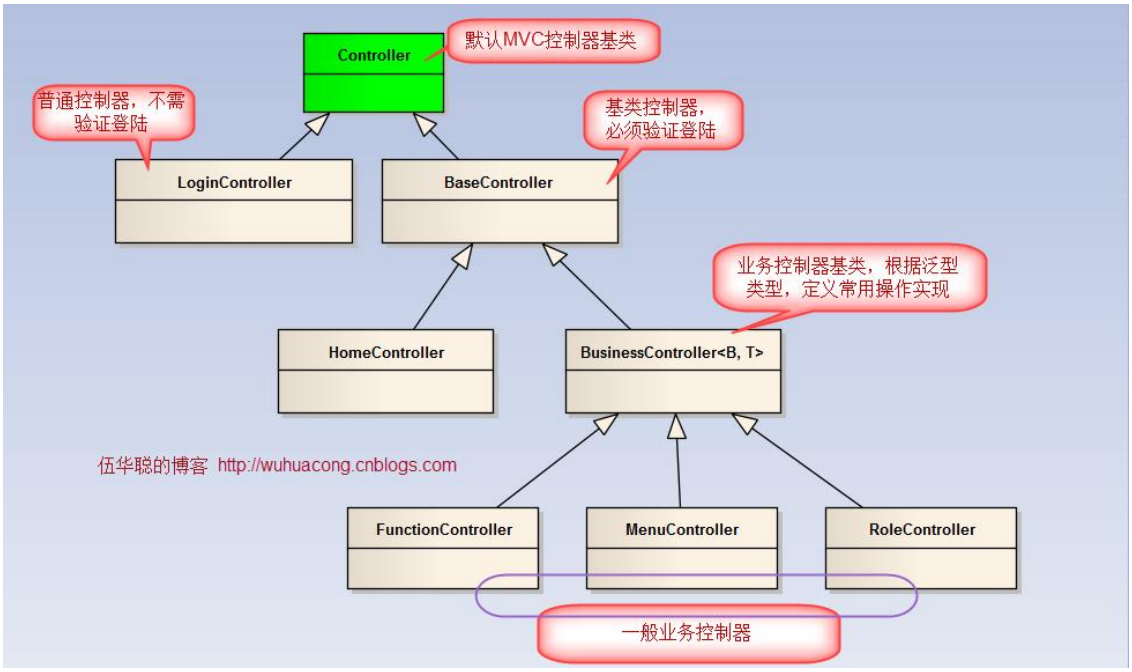
整体框架开发采用 Visual Studuio 2013 / 2017 以及页面编辑工具 Sublime Text 结合开发，页面以及后台代码，通过代码生成工具 Database2Sharp 进行快速开发，实现整体性开发的最大效率提高。

框架的总体结构如下所示：



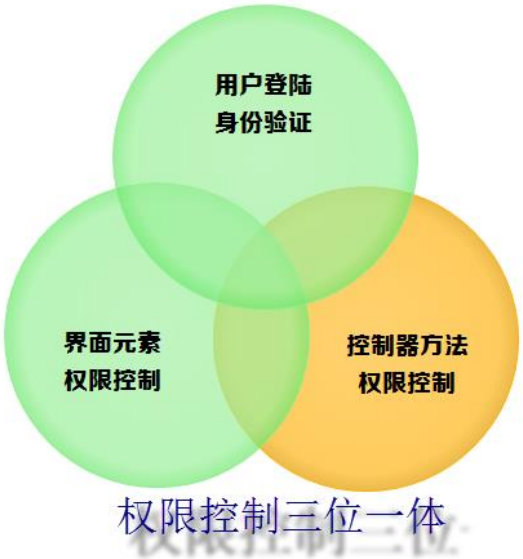
2）控制器设计

Bootstrap 开发框架沿用了我的《Winform 开发框架》和《基于 EasyUI 的 Web 框架》的很多架构设计思路和特点，对 Controller 进行了封装。使得控制器能够获得很好的继承关系，并能以更少的代码，更高效的开发效率，实现 Web 项目的开发工作，整个控制器的设计思路如下所示。



3）权限控制

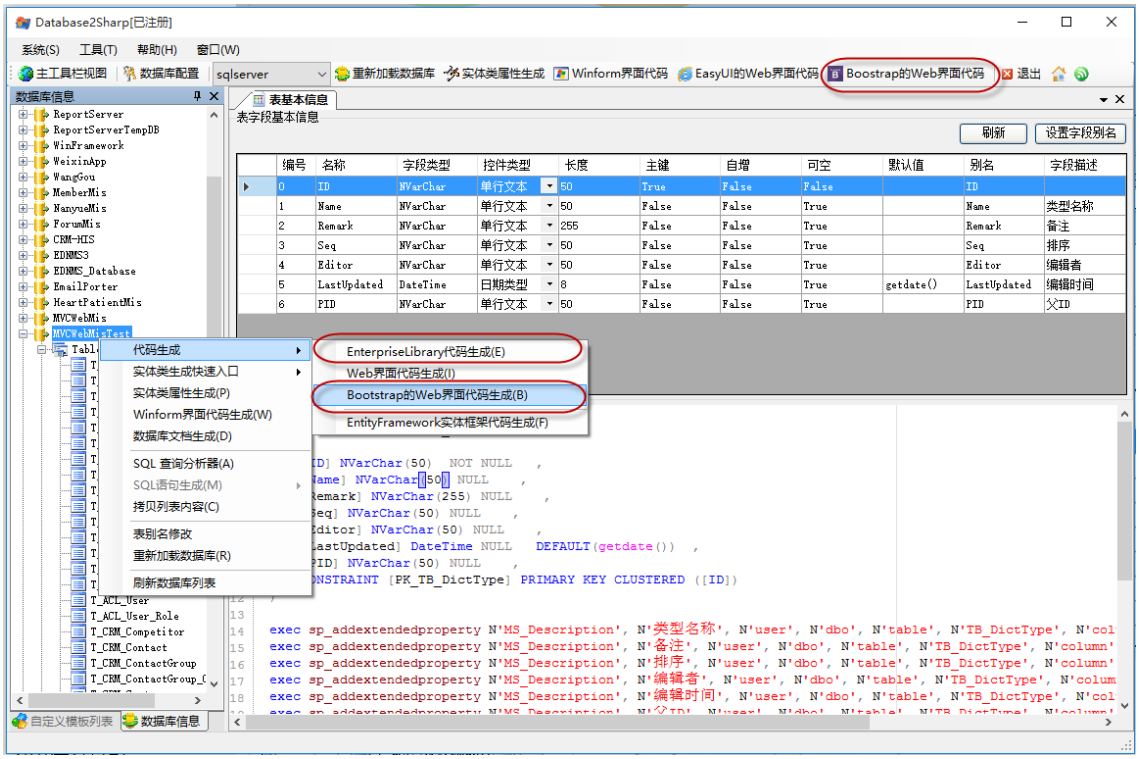
良好的控制器设计规则，可以为 Web 开发框架本身提供了很好用户访问控制和权限控制，使得用户界面呈现菜单、Web 界面的按钮和内容、Action 的提交控制，均能在总体权限功能分配和控制之下。



4）代码快速生成

良好的架构使得无论在业务逻辑层、控制器层、Web 界面的 UI 层，均能提供统一的代码逻辑，这些代码均能通过代码生成工具 Database2Sharp 进行生成。Web 界面代码可以充

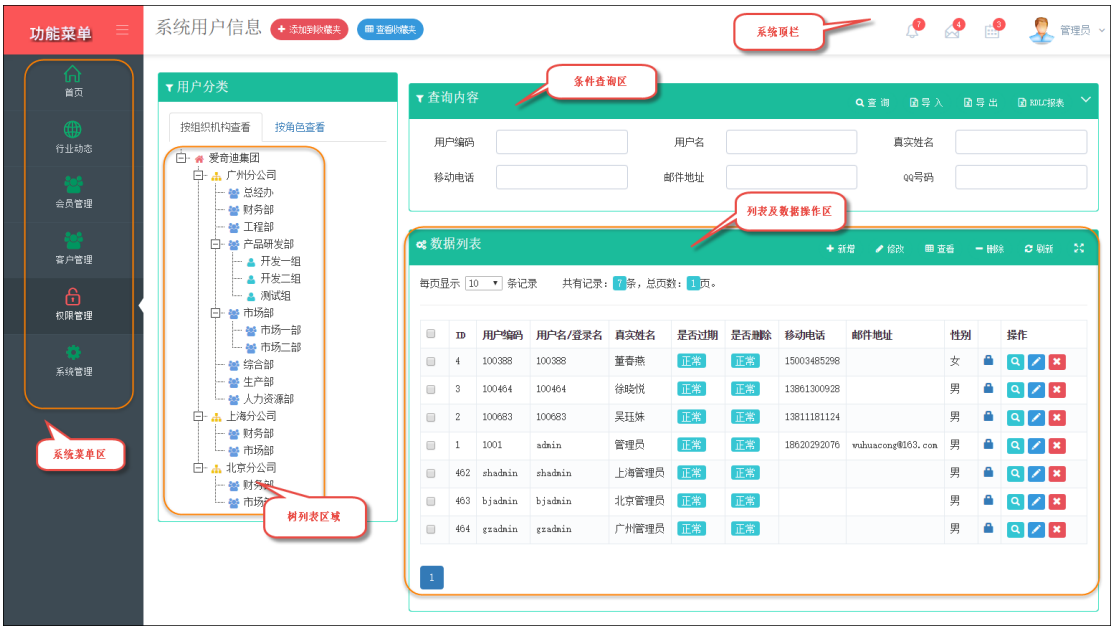
分利用代码生成工具 Database2Sharp 的元数据信息，实现 Web 界面的快速生成。有效减少出错的几率，提高 Web 界面编码的开发效率和乐趣，更可以使得企业内部的编码模式进行高效的统一。



Enterprise Library 代码生成，可以快速生成除界面外的整体性的框架代码，Bootstrap 的 Web 界面代码生成，可以快速生成基于 Metronic 的 Bootstrap 的前端界面代码和后台控制器代码，界面部分包括查询、分页、数据展示、数据导入导出、新增、编辑、查看、删除等基础功能界面，生成后我们可以基于这个基础上进行简单、快速的修改即可符合实际需要，极大提高我们 Web 界面的开发效率。

5）框架布局：

以下是我整体性项目的总的效果图。



【系统菜单栏】的内容，是动态从数据库里面获取的菜单；【系统顶栏】放置一些信息展示，以及提供用户对个人数据快速处理，如查看个人信息、注销、锁屏等操作内容；内容区一般包括【树列表区】、【条件查询区】和【列表数据及分页】内容，内容区域主要是可视化展示的数据，可以通过树列表控件、表格控件进行展示，一般数据还有增删改查、以及分页的需要，因此需要整合各种功能的处理。另外，用户的数据，除了查询展示外，还需要有导入、导出等相关操作，这些是常规性的数据处理功能。

菜单的处理和展示：一般为了管理方便，菜单分为三级，选中的菜单和别的菜单样式有所区分，菜单可以折叠最小化，效果如下所示。

2.2. 登陆及主界面

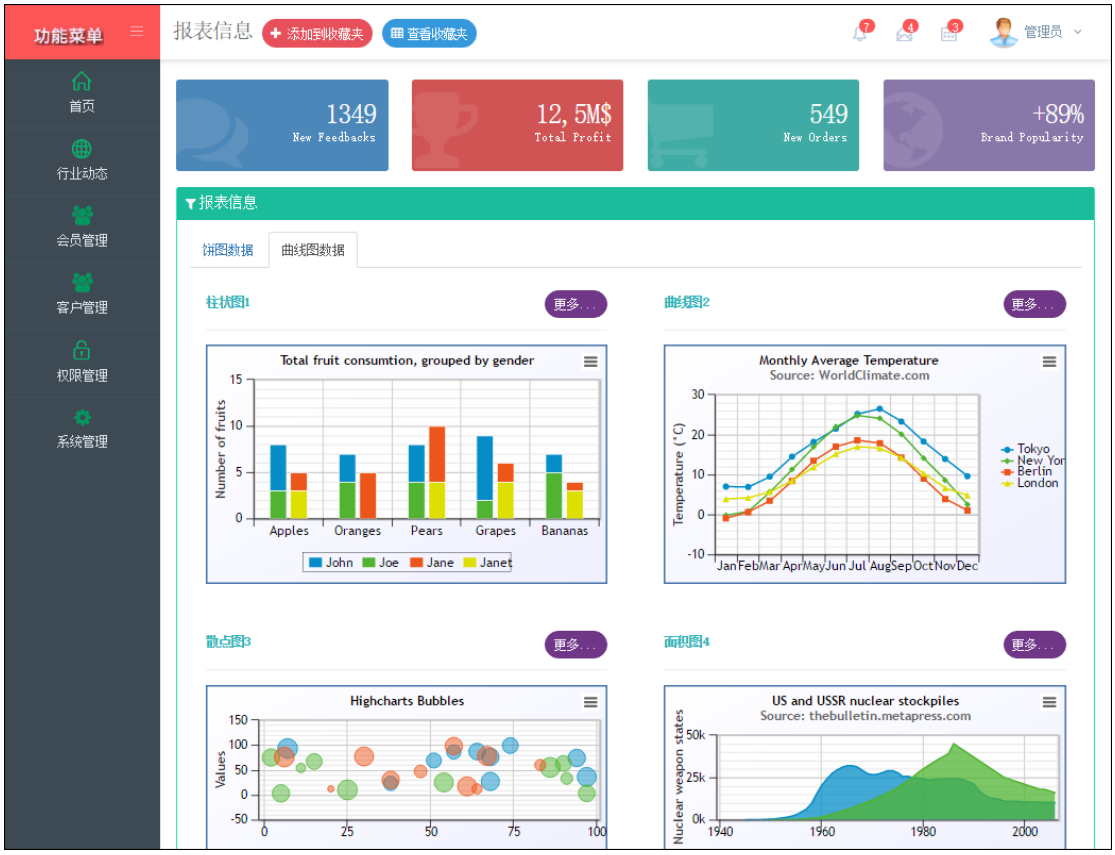
1）登陆界面

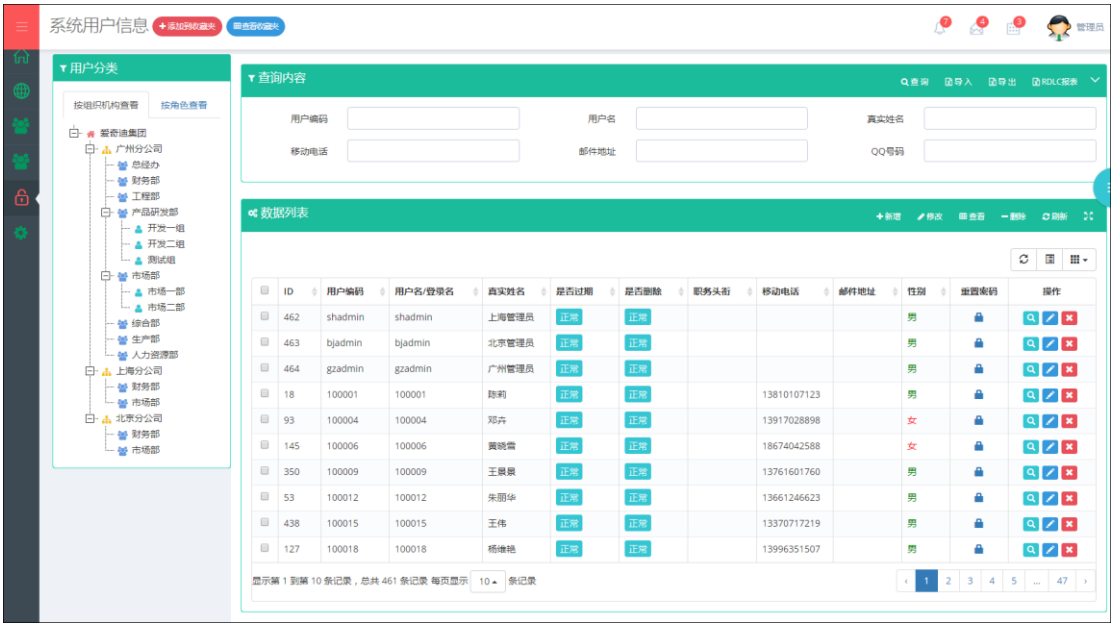
默认登陆界面如下所示：

也可以配置使用其他登陆样式，如下：

在系统登录界面输入账号和密码后（默认账号 admin，密码为空），即可进入管理系统的主界面。

2）框架主体界面


功能菜单
首页
行业动态
会员管理
客户管理
权限管理
系统管理
报表信息
添加到收藏夹
查看收藏夹
管理员
1349
New Feedbacks
12, 5M$
Total Profit
549
New Orders
+89%
Brand Popularity
报表信息
饼图数据
曲线图数据
柱状图1
更多...
Total fruit consumtion, grouped by gender
Number of fruits
Apples
Oranges
Pears
Grapes
Bananas
John
Joe
Jane
Janet
曲线图2
更多...
Monthly Average Temperature
Source: WorldClimate.com
Temperature (°C)
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Tokyo
New Yor
Berlin
London
散点图3
更多...
Highcharts Bubbles
Values
面积图4
更多...
US and USSR nuclear stockpiles
Source: thebulletin.metapress.com
Nuclear weapon states


系统用户信息
用户分类
查询内容
数据列表

3）系统顶栏功能

个人信息-当前用户详细信息：

个人助理-计算器：

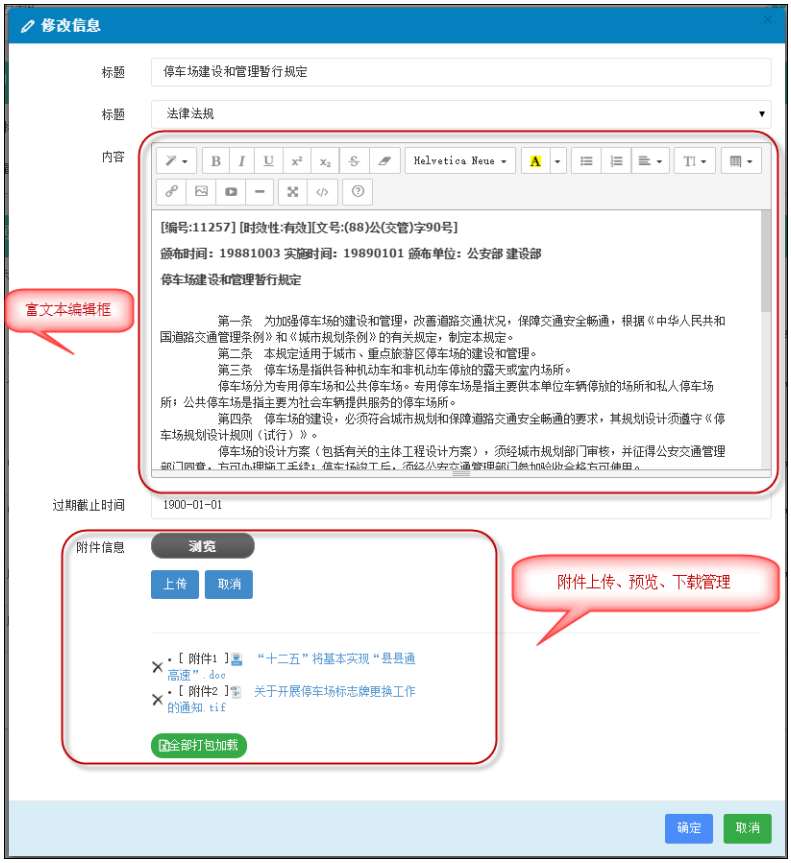
个人助理-公历农历：

锁屏操作：

2.3. 行业动态管理

1）政策法规/通知公告/动态信息 列表界面

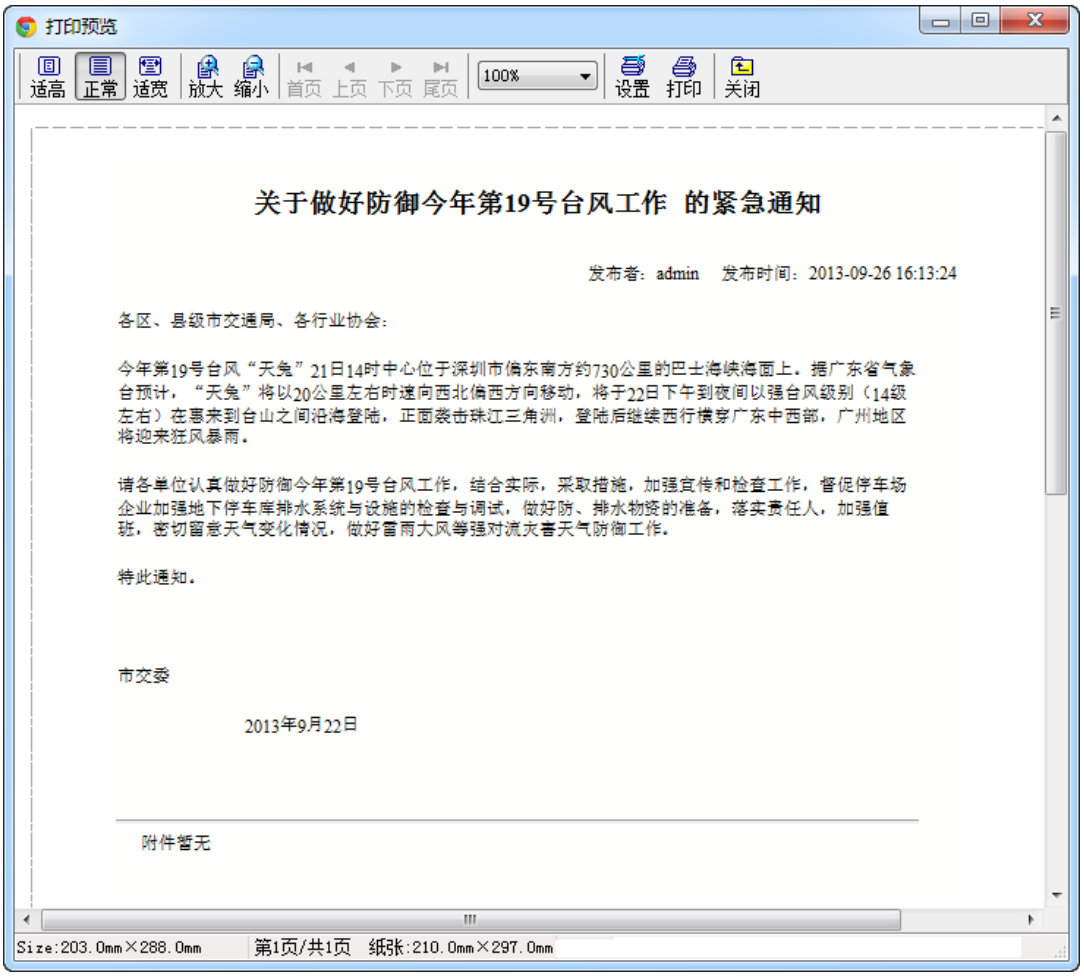
2）编辑界面如下所示：



3）查看内容界面如下所示：

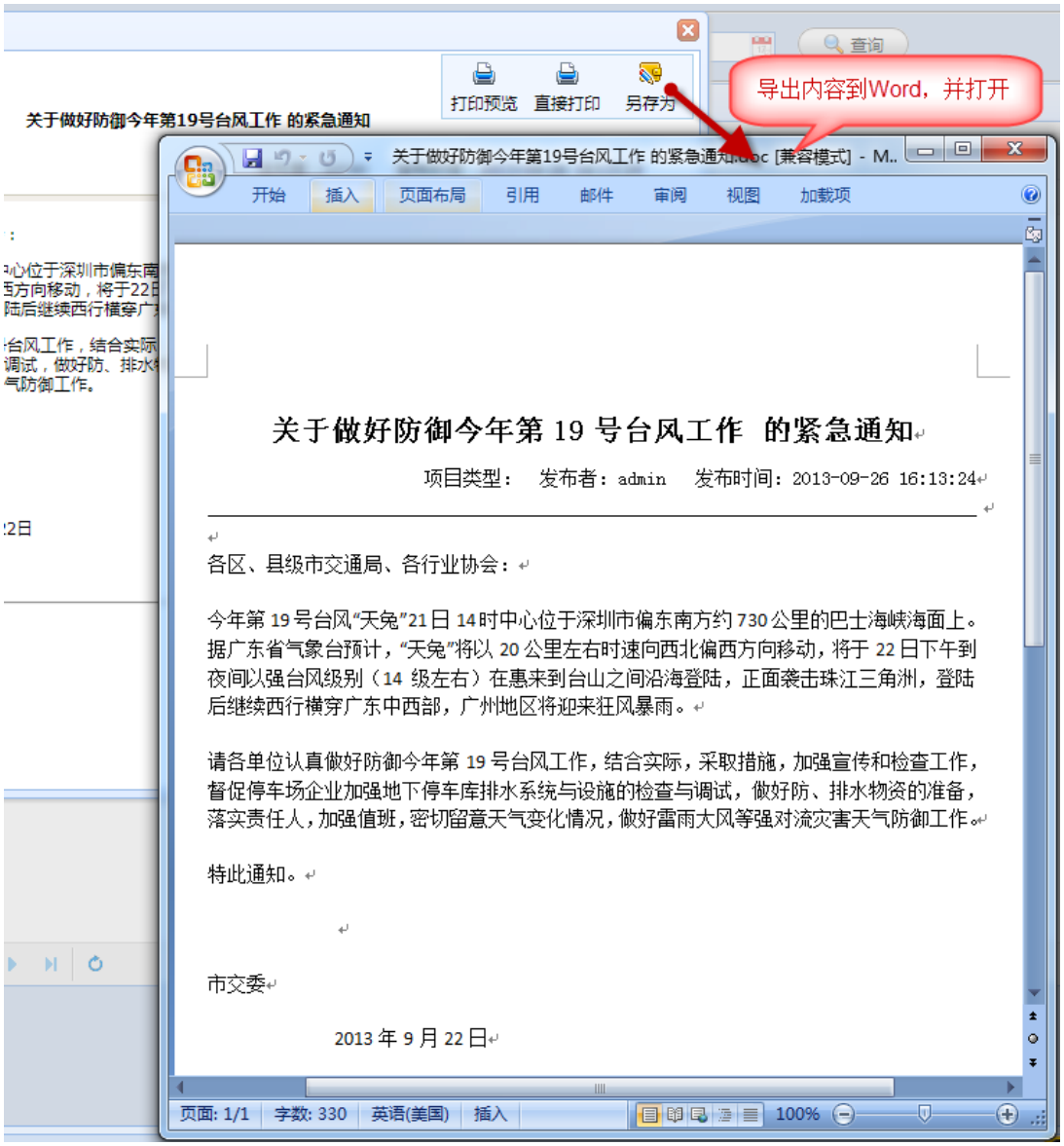
4）打印界面

通知公告内容的打印预览界面如下所示，该模块集成了 LODOP 打印控件，因此预览效果非常美观，这个打印控件在很多场合表现出色，还可以用作证件的套打操作。



5）文件导出到 Word 或者 Excel 操作

通知公告可以导出 Excel 或者 Word 文件，在 MVC 控制器端使用 Aspose.Word 和 Aspose.Excel 控件，通过自定义模板的方式导出 Word 或者 Excel 文件，使得导出的内容更加美观规范。



2.4. 权限系统管理

权限管理模块的主要功能包括有：用户管理、组织机构管理、功能管理、角色管理和权限分配管理、菜单管理、系统类型管理、登录日志管理等功能模块，权限管理系统主界面如下所示。

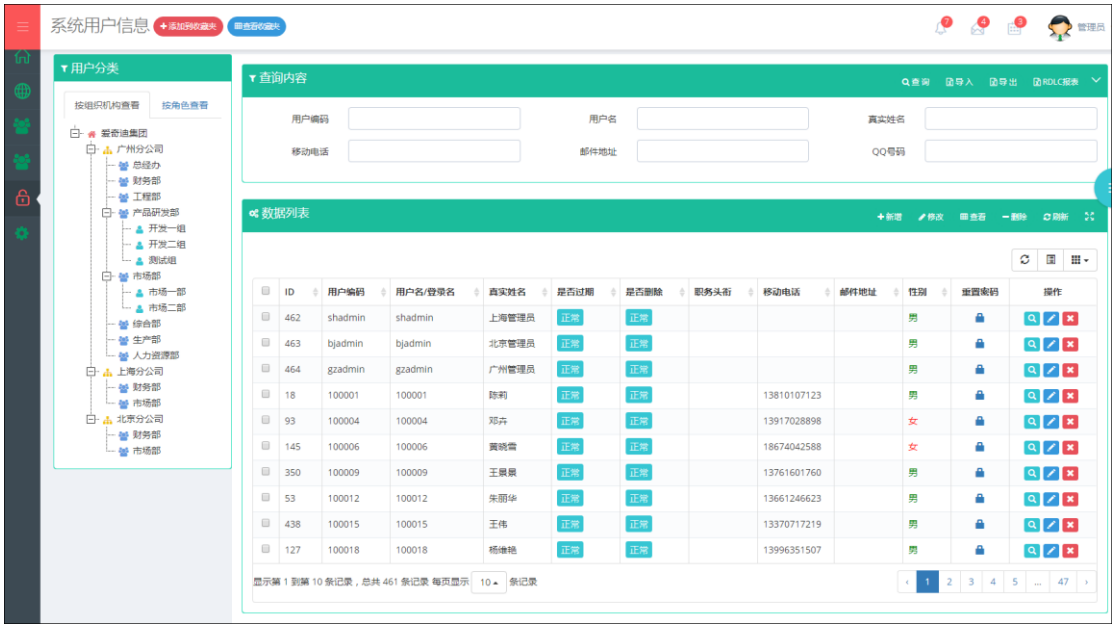
1）用户管理

权限管理系统的用户管理是基于分级管理理念，集团分子公司、事业单位处室/局级可独立管理人员/角色等信息。

系统分了两级管理员用户：超级管理员和公司管理员。超级管理员可以管理整个集团或

者整个系统的人员和相关信息（包括组织机构、角色、登陆日志、操作日志等信息的分级）；公司管理员可以管理分子公司、事业单位处室/局级这样的组织机构的人员和相关信息。

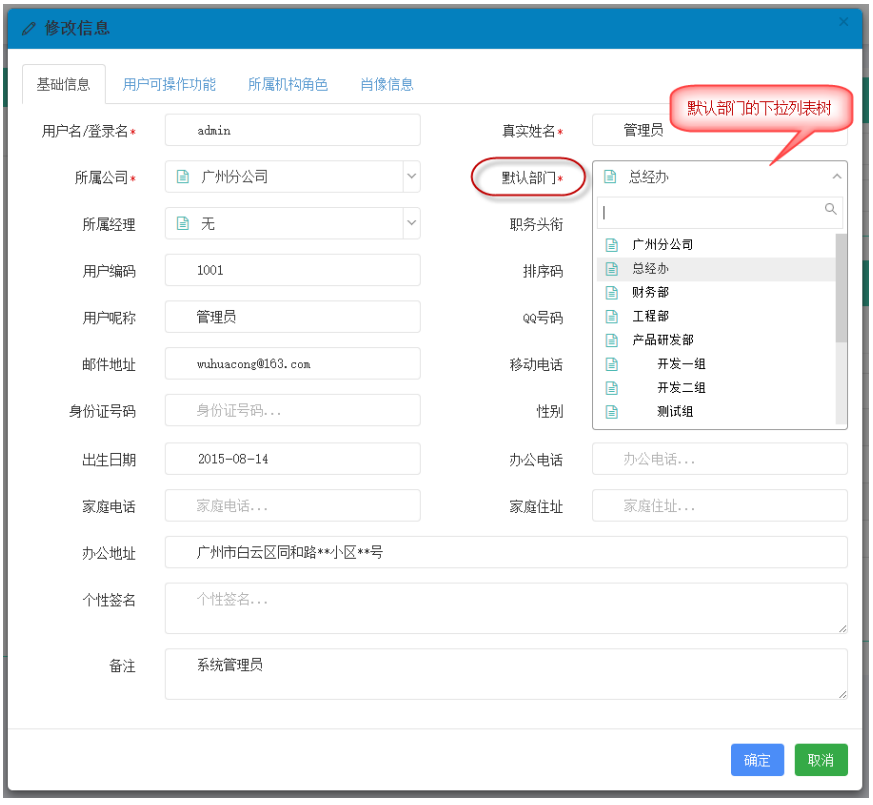
分级管理组织机构、角色、用户等相关数据，能够减少管理员的相关工作，提高工作效率，并能增强权限管理系统对权限的控制和资源分配等管理，提高用户的认同感。



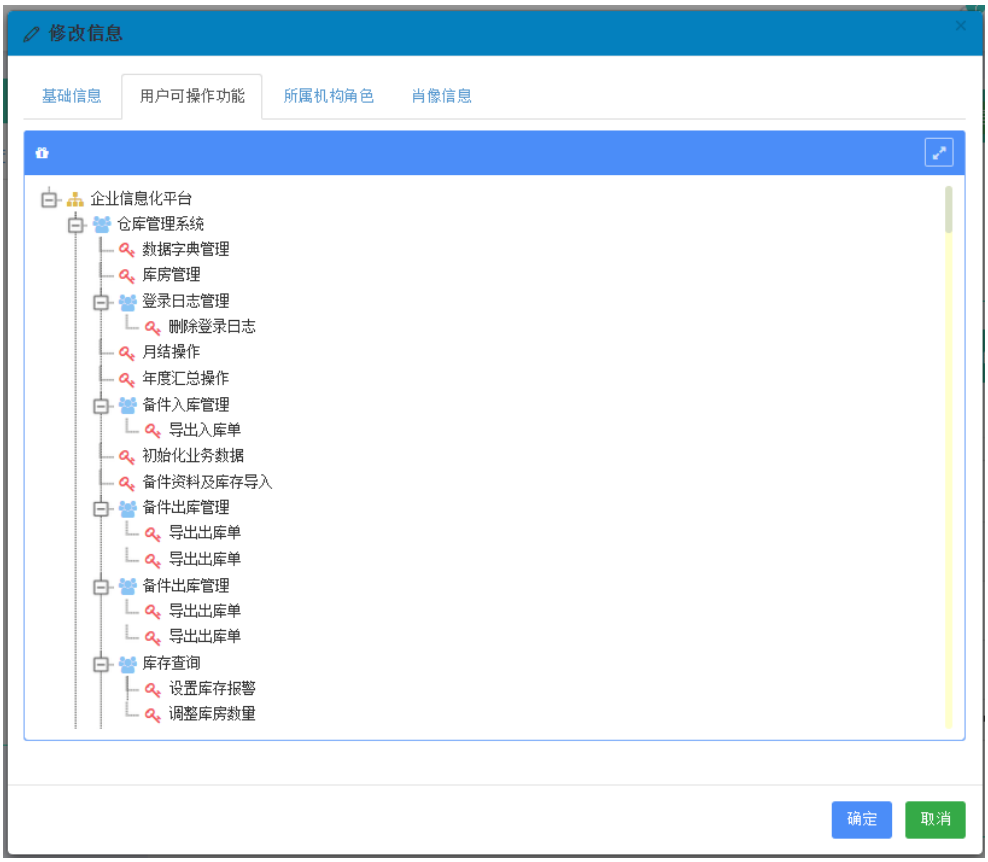
系统用户删除确认对话框。

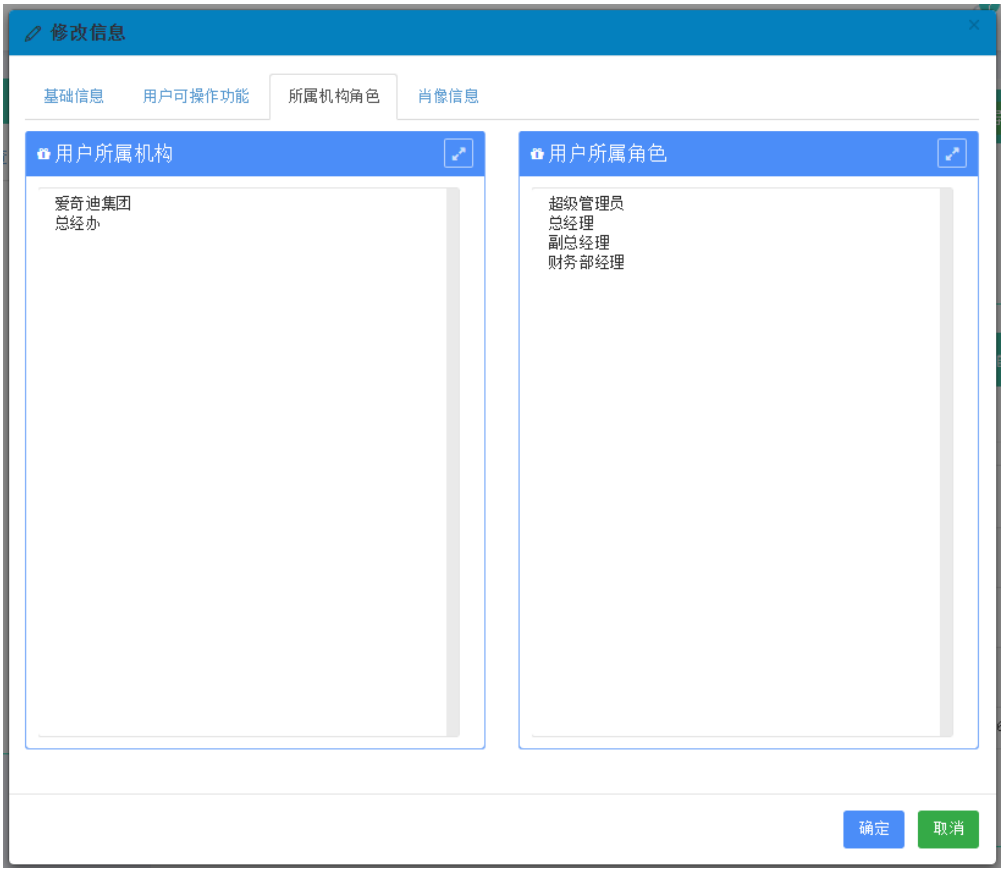
用户分类，可以根据组织结构进行划分，也可以根据角色进行划分，方便查找。

用户信息编辑界面如下所示，包括了用户基础信息和用户可操作功能，可以查看编辑用户的基础信息，也可以查看该用户具有哪些功能。



查看用户可操作功能，是查看该用户包含角色具有的所有功能集合，这里只能进行查看，如果需要调整用户可操作的功能，可在角色管理模块进行权限分配。





系统用户的肖像上传和编辑

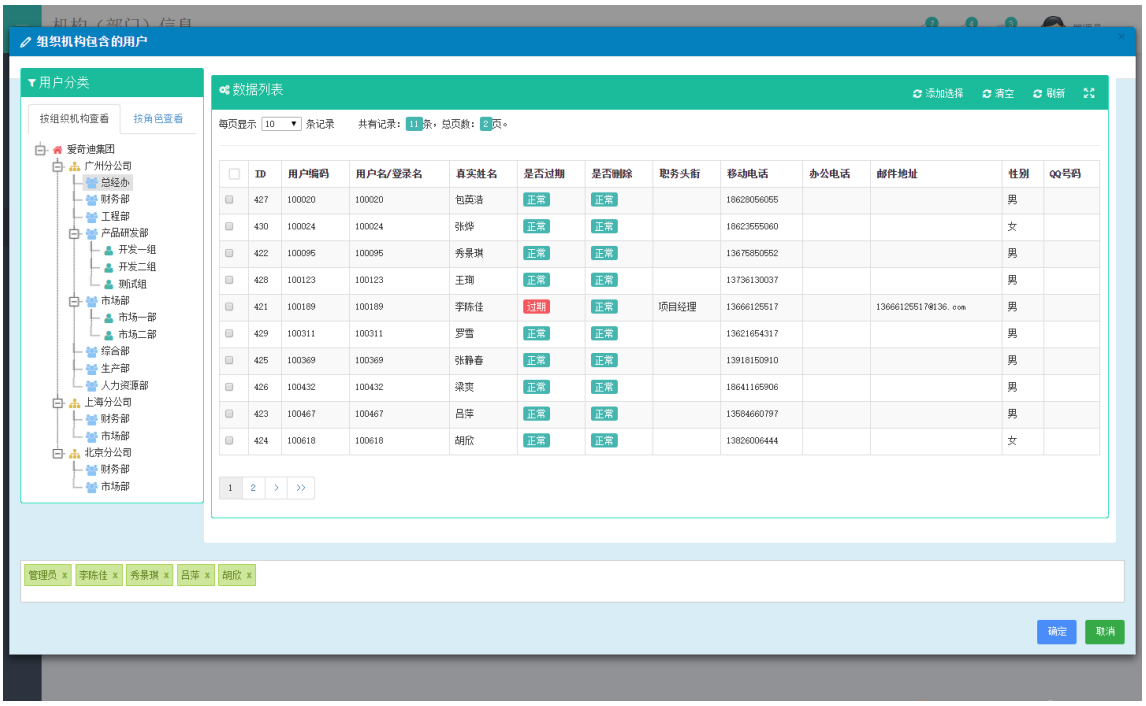
系统用户的 RDLC 报表界面。

2）组织结构管理

为了方便管理，组织机构是以一个树形结构的方式进行展现，组织机构以公司层级进行划分以便实现组织机构的分级管理，每个公司的管理员，只能管理自己公司内部的组织机构关系。

双击任何一个组织机构节点，可以展开机构的详细信息，以及机构的相关信息：包含用户和所属角色。这样可以为组织机构的对应用户，分配具有特定角色，包含人员也就快速具有了对应角色的一切权限。

组织机构包含用户编辑界面



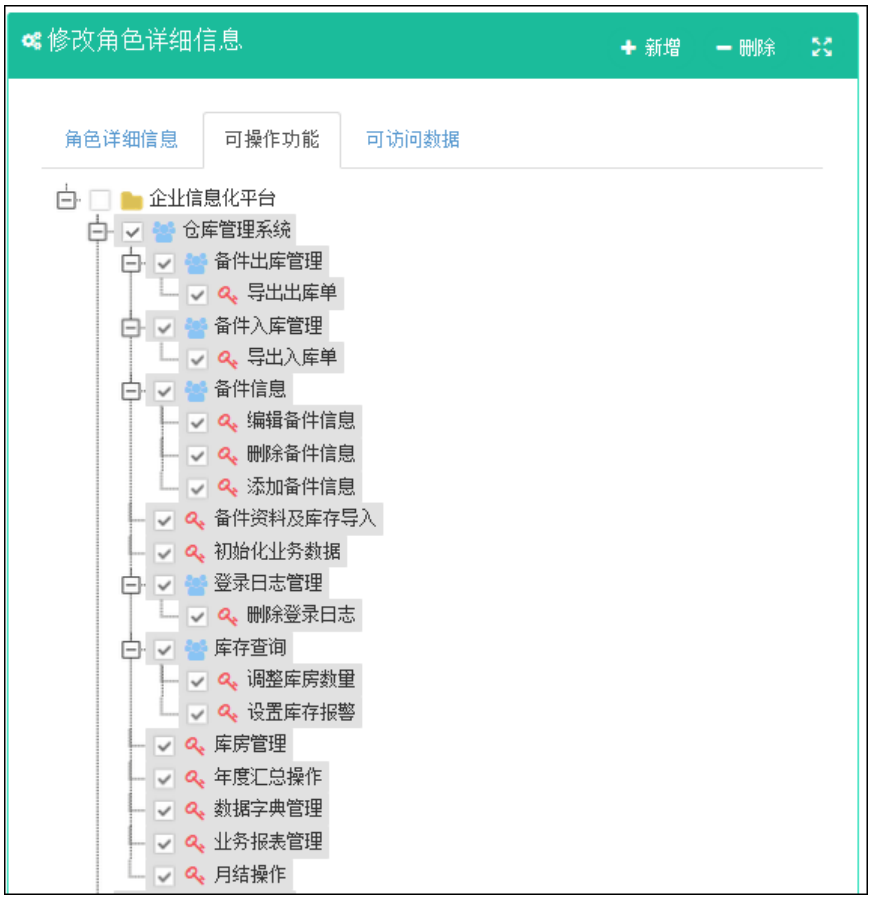
3）角色管理

角色是权限系统管理里面最为重要的部分，整个系统符合权限的国际通用标准，基于RBAC（基于角色的访问控制）的角色权限控制，这样和权限相关的信息，都是通过角色进行关联，因此角色还需要管理和用户之间的关系、和组织机构之间的关系、和可操作功能之间的关系等等。

角色也是根据公司层级进行分级管理的，一个公司内部，角色名称不能重复。角色管理，包括管理角色的基础信息，角色的可操作功能（功能权限）和可访问数据 （数据权限），并通过制定用户或者机构方式，最终实现用户权限的控制。

角色的权限是分级管理的，超级管理员管理所有的角色功能，具有最大的权限集合，可分配不同公司的管理员权限集合；公司所属的管理员，只具有由超级管理员分配的权限。

角色的可操作功能（功能权限），可以在该角色具有的全部权限上分配功能权限。



可访问数据 （数据权限），是通过绑定角色和组织机构关系，从而实现角色数据权限的控制，业务系统在开发过程中进行整合即可有效控制用户的数据权限。


修改角色详细信息
新增
删除
角色详细信息
可操作功能
可访问数据
爱奇迪集团
广州分公司
总经办
财务部
工程部
产品研发部
开发一组
开发二组
测试组
市场部
市场一部
市场二部
综合部
生产部
人力资源部
上海分公司
财务部
市场部
北京分公司
财务部
市场部
保存

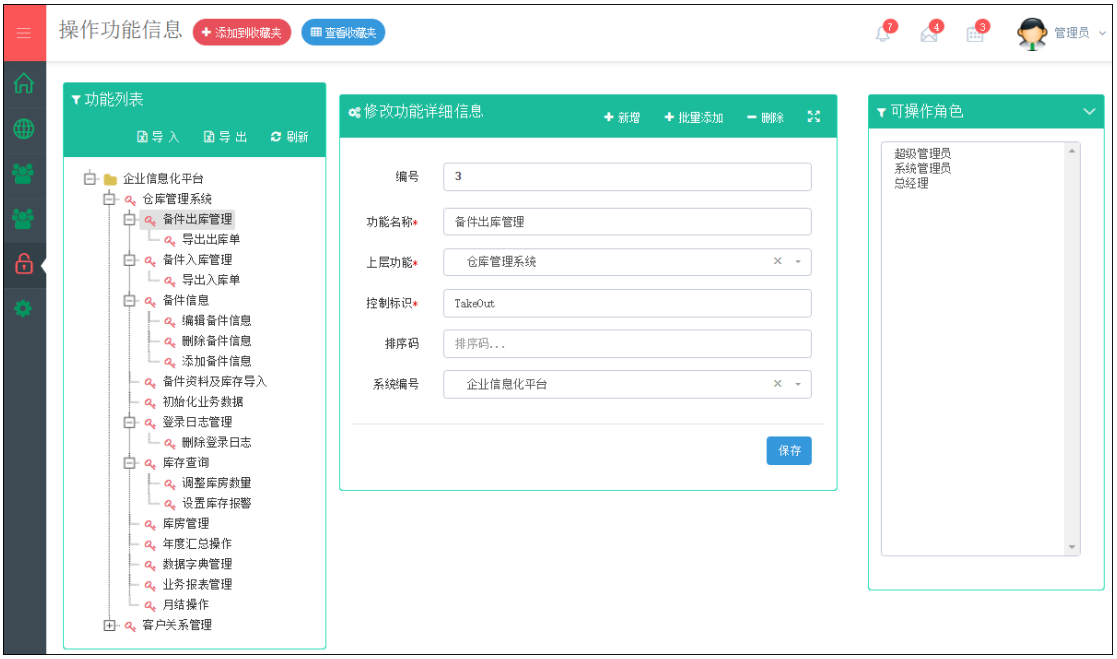
角色包含机构管理：


角色包含的机构
爱奇迪集团
广州分公司
总经办
财务部
工程部
产品研发部
开发一组
开发二组
测试组
市场部
市场一部
市场二部
综合部
生产部
人力资源部
上海分公司
财务部
市场部
北京分公司
财务部
市场部
确定
取消

4）功能管理

功能管理，是业务应用系统的权限控制最小单元，可以用作控制系统的按钮、菜单等界面元素，也可以用作控制显示或隐藏的某些字段的操作。

通过给角色授权不同功能单元，这个角色就具有不同的权限集合。用户的可访问的功能，是通过角色获得的，也就是基于角色的权限控制（RBAC）的原理。

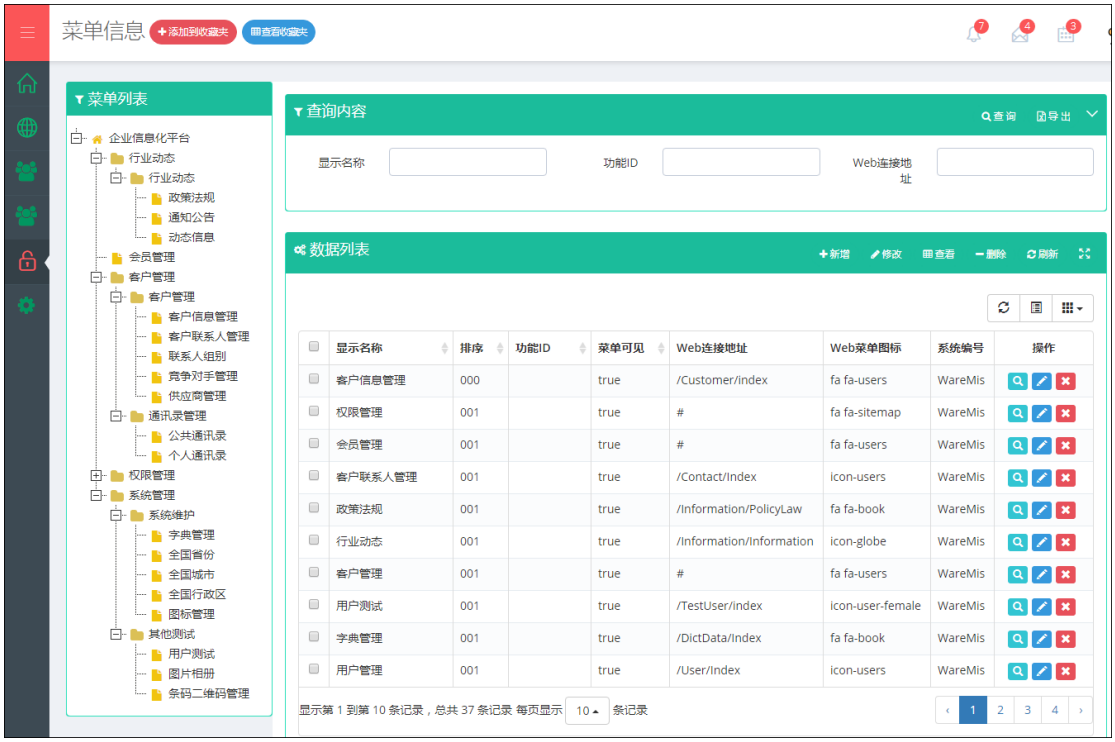


为了方便一次性添加多个功能单元，可以通过“批量添加”操作进行功能的批量添加，批量添加界面如下所示。

5）菜单管理

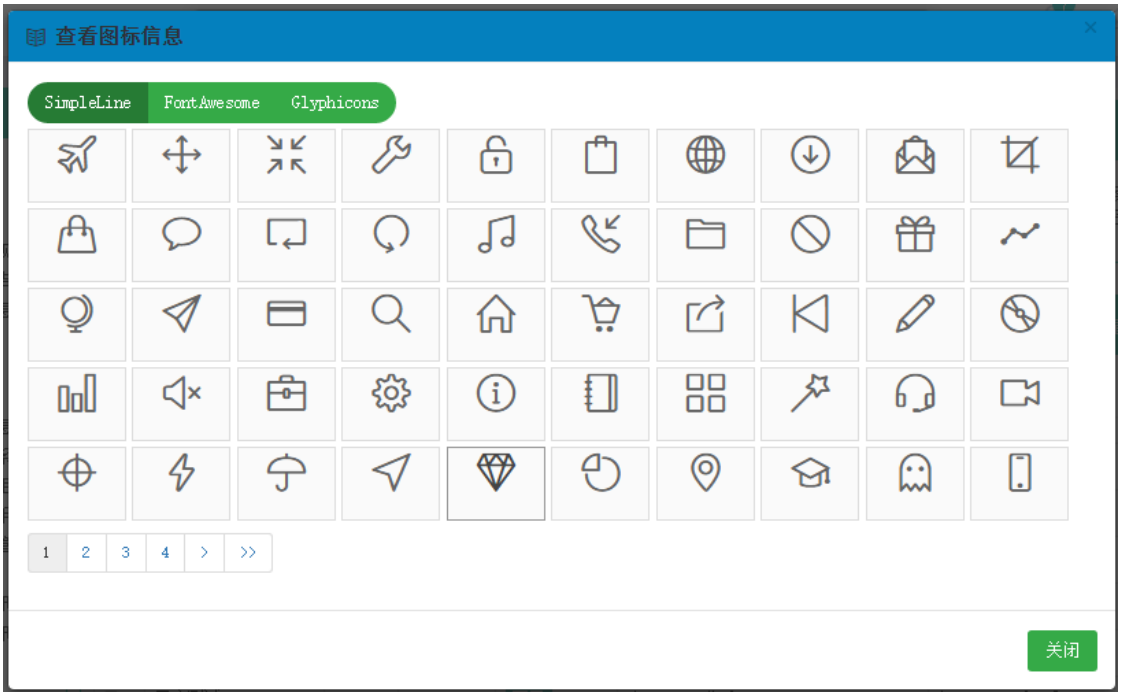
菜单是在基于插件化的系统应用的时候，方便使用来动态创建的，它的权限是通过功能控制 ID 来判断，也就是哪些用户具有这些功能控制 ID，那么就可以访问，否则不显示。一旦用户在访问的时候，父菜单不具有访问权限，那么所有子菜单也不能访问。

因此菜单也是权限分配的一部分，为了有效管理菜单资源，我们把菜单放到权限管理系统中进行管理控制，可根据用户权限进行动态控制显示。

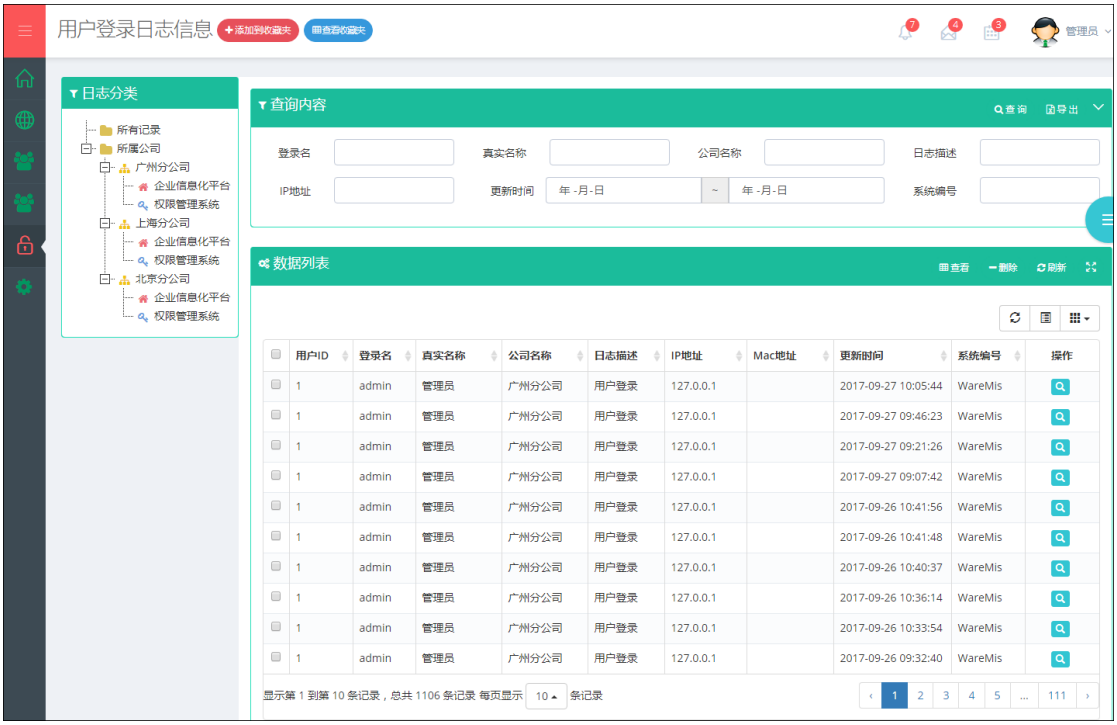


编辑菜单界面如下所示。

选择菜单图标界面：

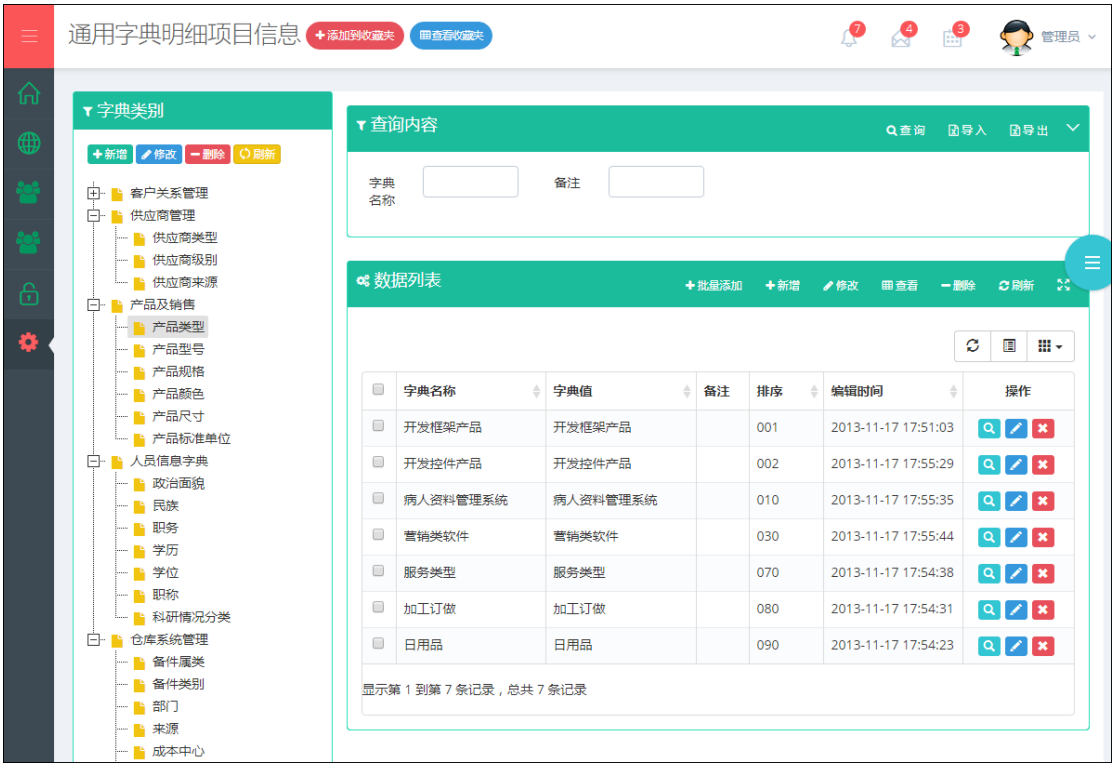

查看图标信息
SimpleLine
FontAwesome
Glyphicons
1
2
3
4
关闭

6）登陆日志


用户登录日志信息
日志分类
查询内容
数据列表
登录名
真实名称
公司名称
日志描述
IP地址
更新时间
系统编号
用户ID
登录名
真实名称
公司名称
日志描述
IP地址
Mac地址
更新时间
系统编号
操作

2.5. 字典管理

字典管理包括了字典类别的管理和字典数据的管理，通过在界面中集成树控件，单击任何一个节点，均能进行类别字典数据的刷新，然后在右边进行展示，界面如下所示。

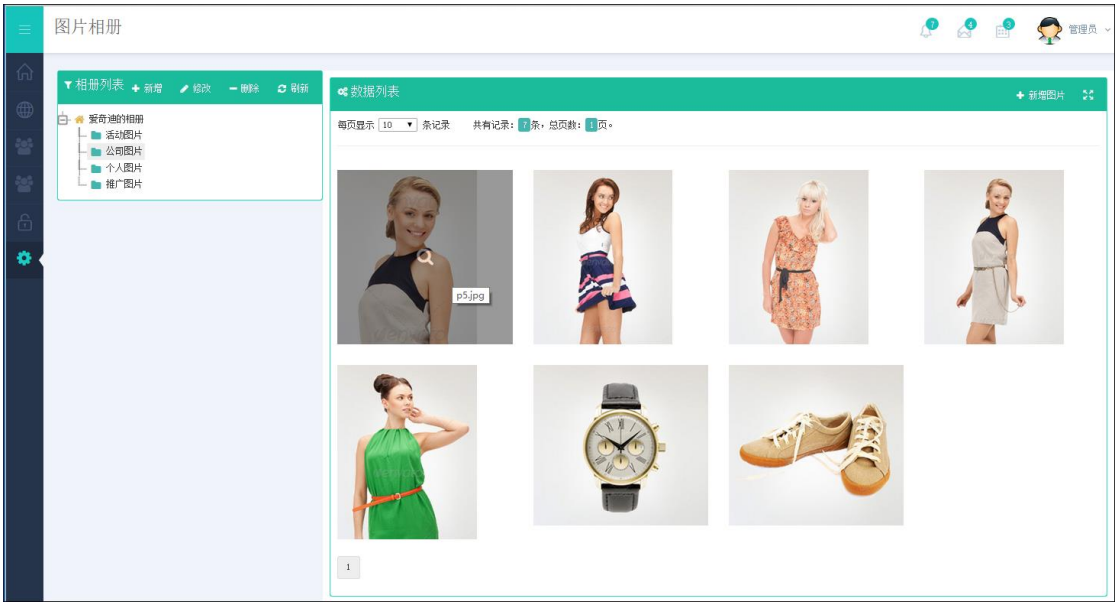


在字典类别中添加，弹出一个对话框，并以当前的字典类别作为父类节点，界面如下所示。

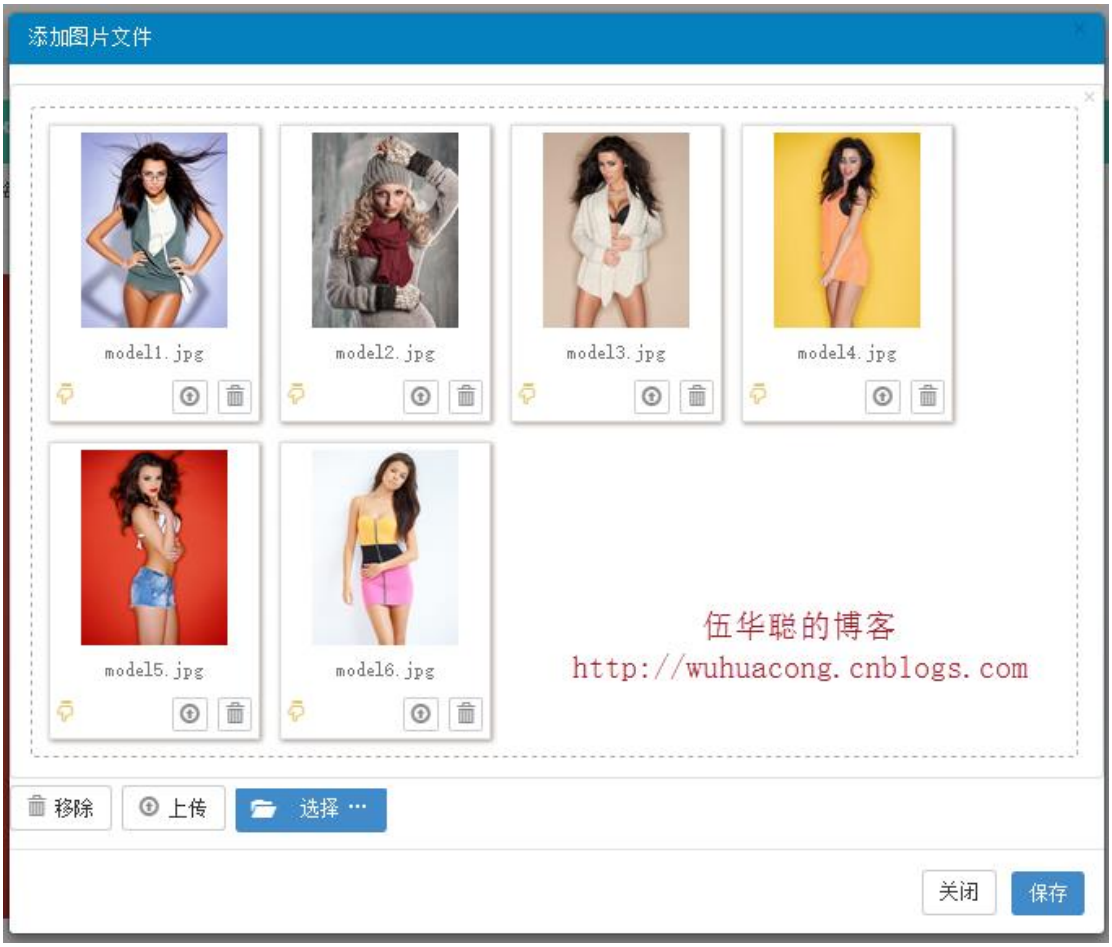
修改字典数据的界面如下所示。

除了单项字典数据的添加，有时候，批量添加字典数据也是很方便、很重要的，因此提供了一个批量字典数据录入的界面，如下所示。

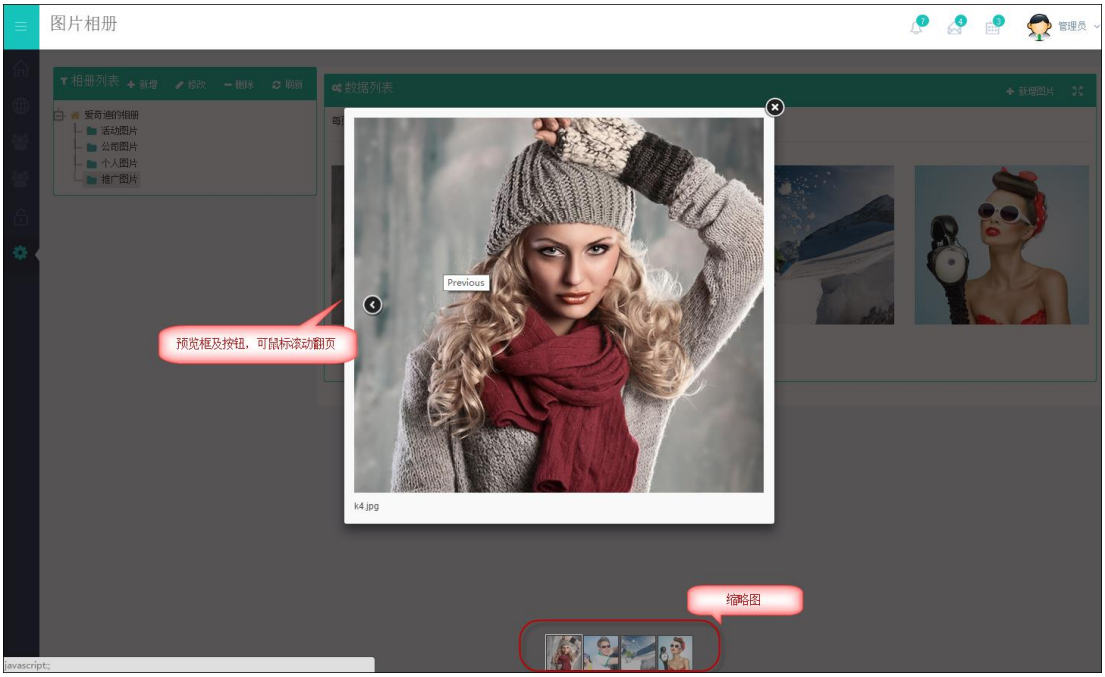
2.6. 图片相册管理



图片编辑界面如下所示：



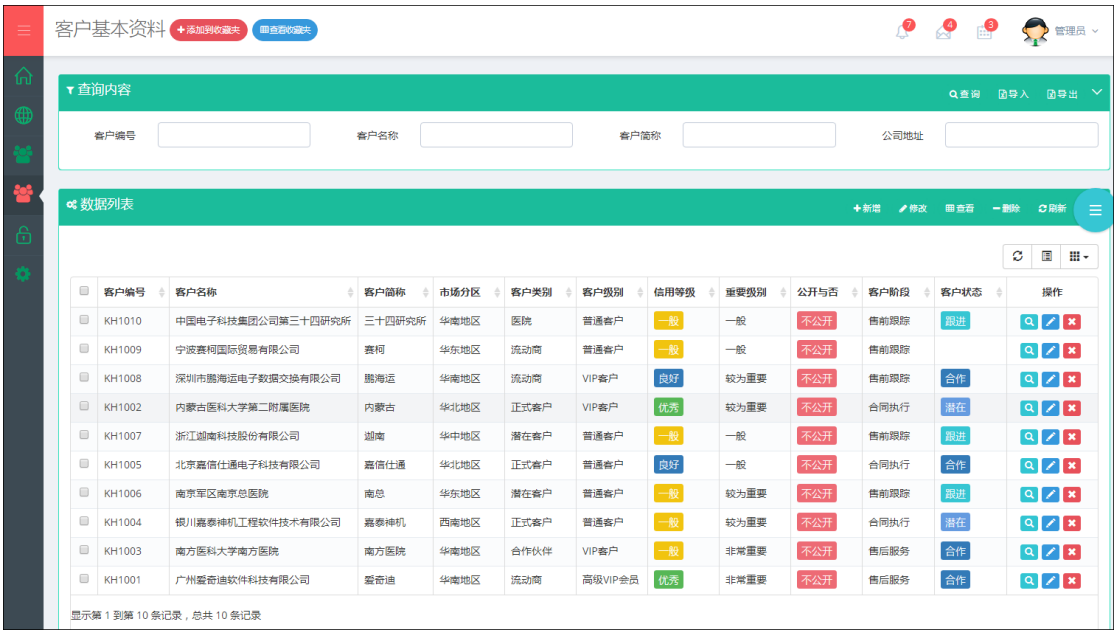
图片查看界面



2.7. 客户关系管理

2.7.1. 客户信息管理

客户列表界面如下所示：



客户信息编辑界面：


修改信息
Tab选项卡展示
客户基础信息
客户相关信息
客户状态信息
客户编号
KH1001
客户名称
广州爱奇迪软件科技有限公司
客户简称
爱奇迪
所在省份
广东省
城市
广州市
所在行政区
白云区
市场分区
华南地区
公司地址
数据关联刷新
公司邮编
510515
办公电话
87207160
传真号码
传真号码...
主联系人
伍华聪
联系人电话
18620292076
联系人手机
18620292076
电子邮件
wuhuacong@163.com
QQ号码
6966254
所属行业
信息传输、计算机服务和软…
客户来源
朋友介绍
客户类别
流动商
客户级别
高级VIP会员
信用等级
优秀
重要级别
非常重要
公开与否
公开
下拉列表，字典数据
客户阶段
售后服务
客户状态
合作
备注信息
备注信息...
确定
取消

客户信息导入界面：


文件导入
标准模板，单击可下载
Customer-模板.xls
浏览
上传
取消
上传文件并预览列表
客户编号
客户名称
客户简称
所在省份
城市
所在行政区
市场分区
客户类别
客户级别
信用等级
重要级别
公开与否
备注信息
客户阶段
客户状态
勾选复选框并保存
关闭
保存

2.7.2. 客户联系人管理

客户联系人添加/编辑界面

客户联系人查看界面


查看信息
基础信息
相关信息
附件信息
客户名称
广州爱奇迪软件科技有限公司
编号
1001
排序序号
1001
姓名
伍华聪
身份证号码
123456
出生日期
1900-01-01 00:00:00
性别
男
办公电话
87079089
家庭电话
传真
87079089
联系人手机
18620292076
联系人地址
广州市白云区
邮政编码
510515
电子邮件
wuhuacong@163.com
QQ号码
6966254
备注
重要级别
非常重要
认可程度
忠诚度高
关系
紧密
负责需求
推广服务,维护服务
关心重点
安全,产品外观形象
利益诉求
满足需求,社会影响
关闭

附件信息界面

2.8. 通讯录管理

通讯录在很多系统均有使用，是一个非常常见的功能模块，包含了公共通讯录和个人通讯录，公共通讯录一般是指公司内部的通讯录，由专人维护；个人通讯录是当前用户自己的通讯录，用户负责自己的个人通讯录维护工作。

2.8.1. 公共通讯录

通讯录编辑界面如下所示。

通讯录分组管理界面如下所示：

2.8.2. 个人通讯录

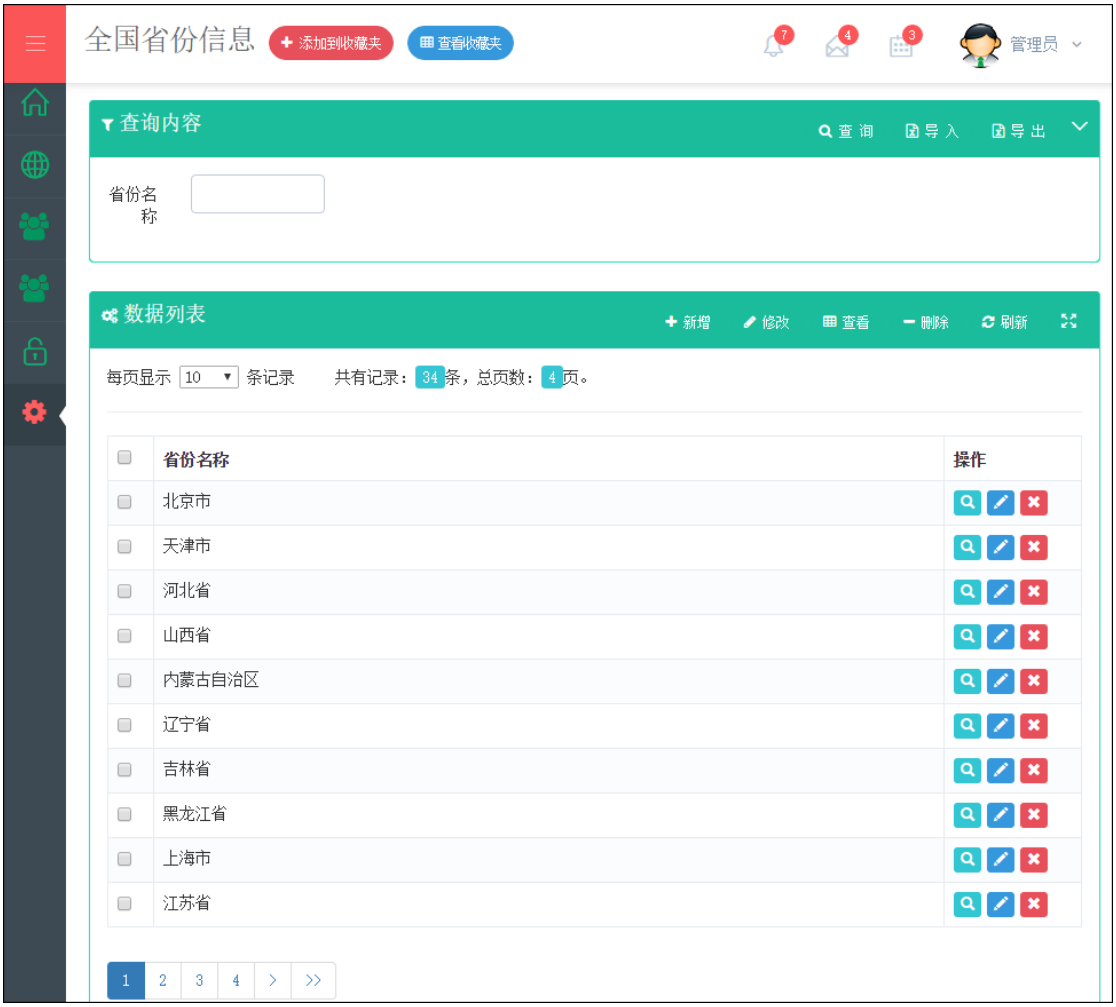
2.9. 全国行政区划管理

在最近的项目中，由于需要使用到全国地区的省份、城市、区县这些新政区划的信息，网上的很多数据脚本都是早期的，因此花了一些时间，重新校对了这些省份、城市、地区的数据内容，以及编写一个对全国省市地区的数据字典管理模块，由于这些模块的数据是属于

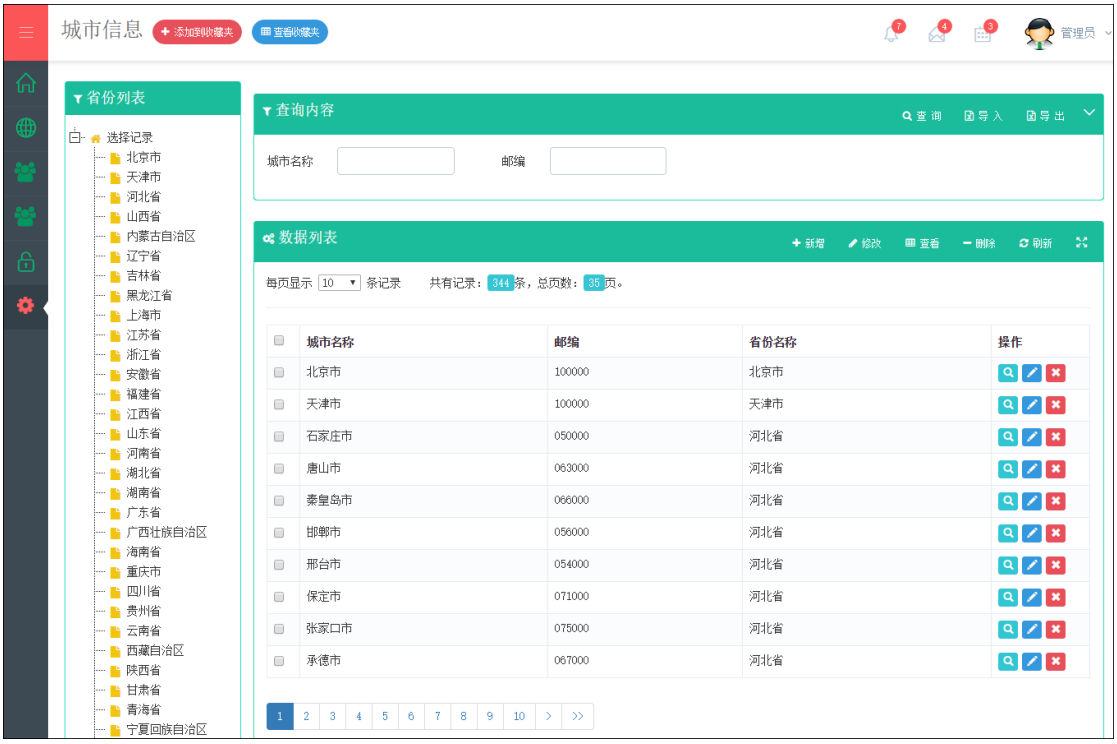
数据字典的范畴，因此把这些新政区划的管理也整合通用字典模块里面，以方便更好的应用。

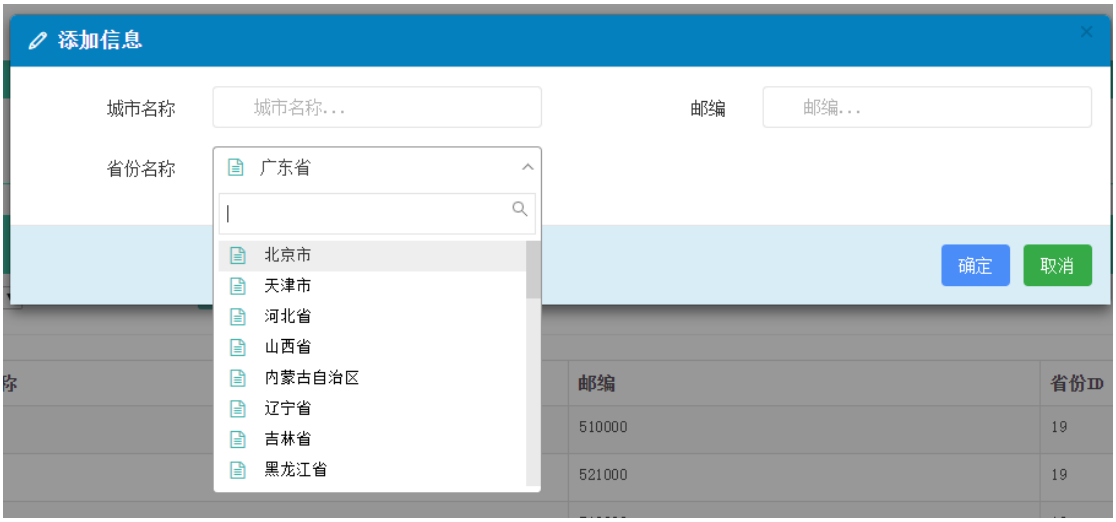
这些全国的新政区划数据，最权威的数据当然来自国家统计局了（http://www.stats.gov.cn/tjbz/xzqhdm/t20130118_402867249.htm），从里面的数据可以看出，这两三年，全国的省份信息基本没有变化，但是城市、区县的数据变化还是不少，如调整了三亚，增加了三沙市，以及合并一些省份的城市，广州等城市的区县也有所变化等等，因此花了不少精力时间来对这些数据进行整理，希望能够给自己方便使用的同时，也方便需要用到这些省份城市行政区的开发人员。

2.9.1. 省份管理

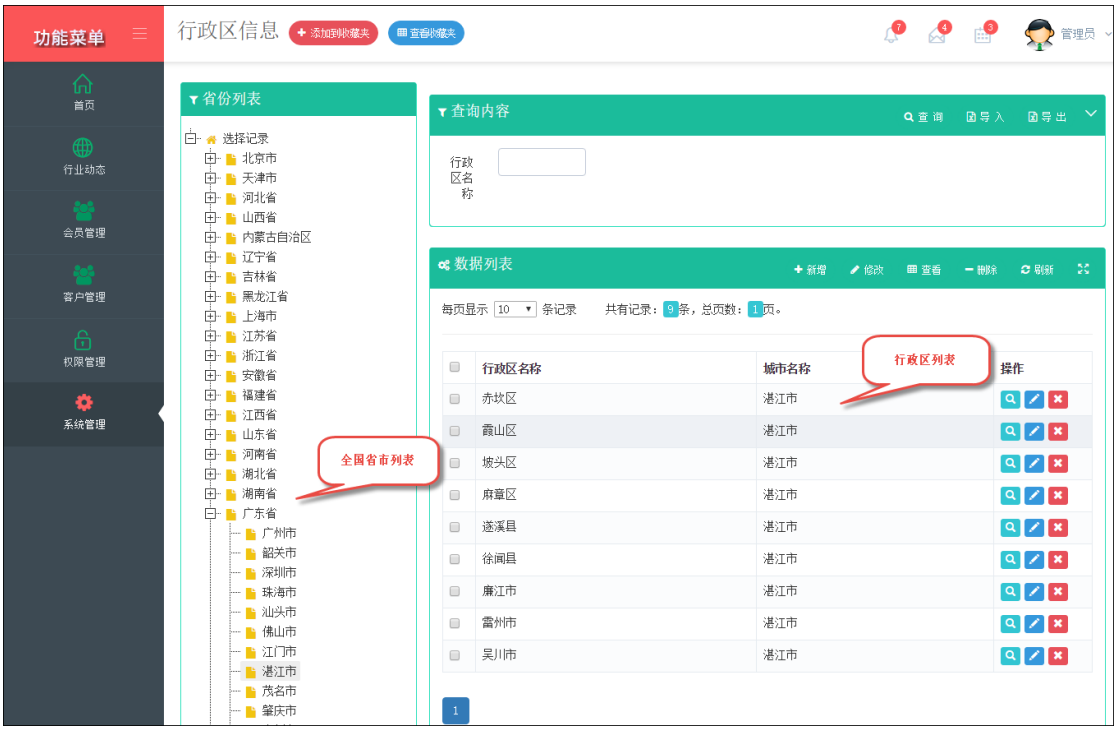


2.9.2. 城市管理


城市信息
添加城市信息
查看城市信息
管理员
省份列表
选择记录
北京市
天津市
河北省
山西省
内蒙古自治区
辽宁省
吉林省
黑龙江省
上海市
江苏省
浙江省
安徽省
福建省
江西省
山东省
河南省
湖北省
湖南省
广东省
广西壮族自治区
海南省
重庆市
四川省
贵州省
云南省
西藏自治区
陕西省
甘肃省
青海省
宁夏回族自治区
查询内容
查询
导入
导出
城市名称
邮编
数据列表
新增
修改
查看
删除
刷新
每页显示 10 条记录
共有记录：344条，总页数：35页。
城市名称
邮编
省份名称
操作
北京市 100000 北京市
天津市 100000 天津市
石家庄市 050000 河北省
唐山市 063000 河北省
秦皇岛市 066000 河北省
邯郸市 056000 河北省
邢台市 054000 河北省
保定市 071000 河北省
张家口市 075000 河北省
承德市 067000 河北省
1 2 3 4 5 6 7 8 9 10 > >>


添加信息
城市名称
城市名称...
邮编
邮编...
省份名称
广东省
北京市
天津市
河北省
山西省
内蒙古自治区
辽宁省
吉林省
黑龙江省
确定
取消
邮编
省份ID
510000
19
521000
19

2.9.3. 行政区管理



有了这些全国省份、城市和行政区的数据，我们就可以在界面上做相应的关联，联动获得对应的城市信息了，如下面截图所示。

2.10. 页面链接收藏夹功能

在一个系统里面，往往有很多菜单项目，每个菜单项对应一个页面，一般用户只需要用到一些常用的功能，如果每次都需要去各个层次的菜单里面去找对应的功能，那确实有点繁琐。特别是在菜单繁多，而客户又对系统整体不熟悉的情况下，如果有一个类似浏览器的收藏夹模块，把一些常用的菜单连接保存起来，每次从这个收藏夹主页去找对应的页面，那样确实是省事省力，非常方便。

为了实现这个收藏夹功能，我们也需要在系统页面的明显位置处放置一个收藏夹模块的入口，以及可以为每个页面添加到对应收藏夹的功能。经过对比，我们把这些入口功能放在页面标题的附近，这样方便进行快速进行收藏夹，如下效果所示。

当我们在页面上单击【添加到收藏夹】按钮，我们就把对应的页面标题和连接加入到收藏夹记录里面了。

在【查看收藏夹】功能里面，我们可以展示我们加入的页面链接，单击其中某个记录，可以快速进入对应的页面，这样就实现了我们快速进入功能模块的需求了。

另外我们可以收藏夹列表里面进行记录的删除、拖动操作，通过鼠标的拖放就可以实现位置的上下移动，非常方便。

2.11. 条码和二维码的生成打印处理

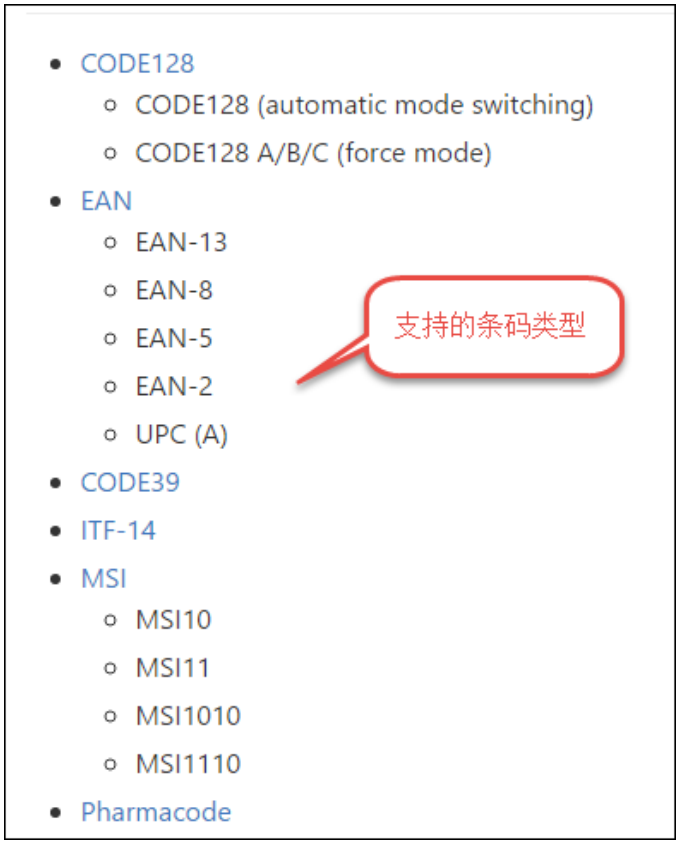
在很多项目里面，对条形码和二维码的生成和打印也是一种很常见的操作，在 Web 项

目里面，我们可以利用 JS 生成条形码和二维码的组件有很多。本框架引入两个比较广泛使用的 JS 组件，用来处理条形码和二维码的生成处理，并利用 CLODOP 组件实现内容的打印输出。生成条形码使用组件 JsBarcode，生成二维码使用组件 qrcodejs。

2.11.1. 条形码的生成

条码的作用一般在一些商品标签上，方便使用条码枪快速、准确录入信息。如下所示是一种条形码。

这里条形码生成使用了 JsBarcode 组件进行处理，它支持很多格式的条码格式，如下所示。

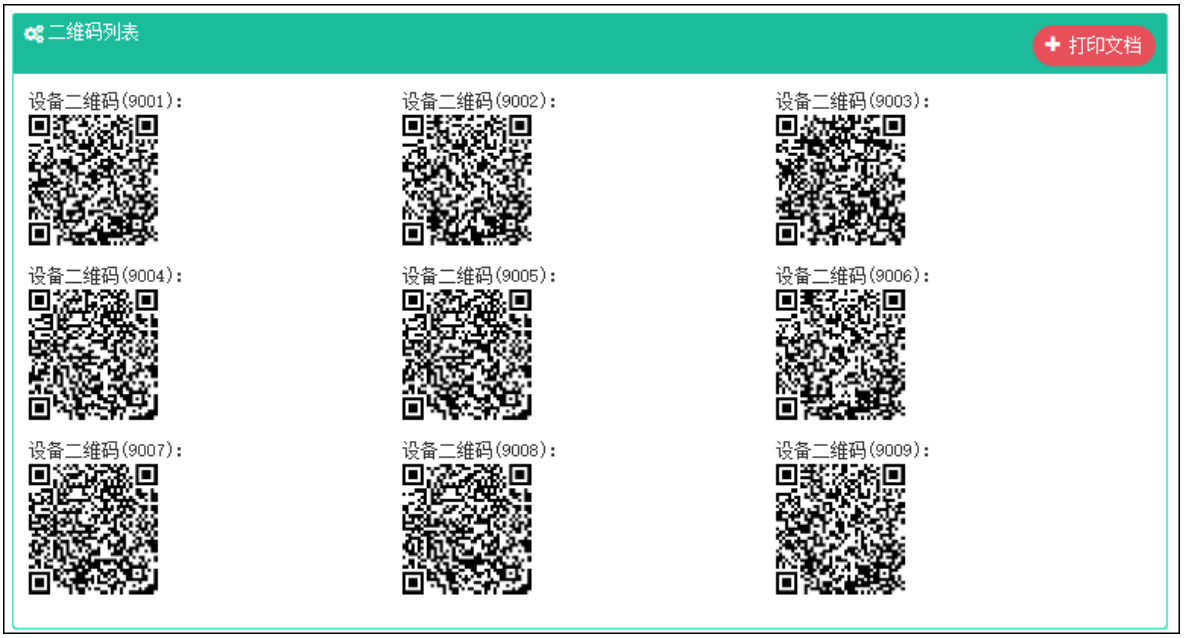


具体的生成效果如下所示。

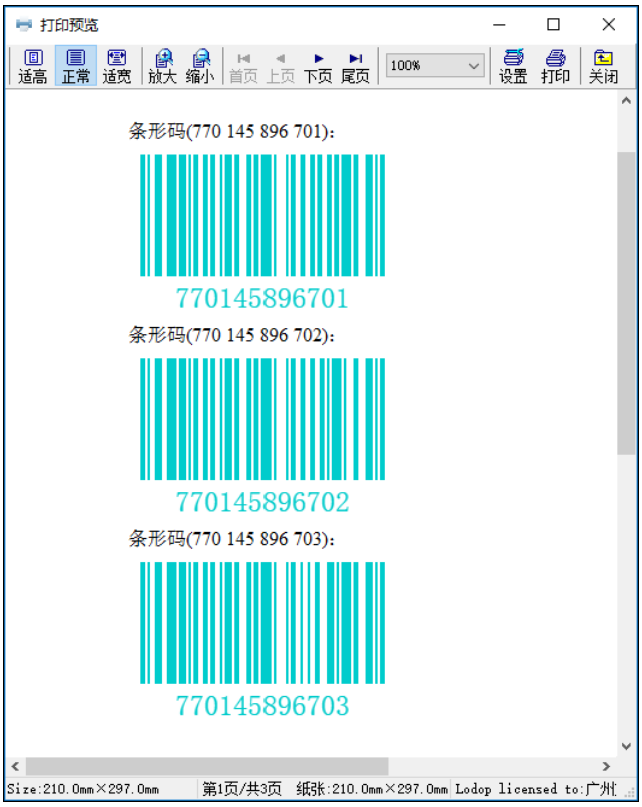
2.11.2. 二维码的生成

二维码实现可以通过使用组件 qrcodejs 进行生成，二维码也可以使用组件 jquery-qrcode 进行生成，也相对比较简洁，不过打印二维码文档的时候，jquery-qrcode 没有显示二维码图片，而组件 qrcodejs 则工作正常，因此推荐使用组件 qrcodejs。

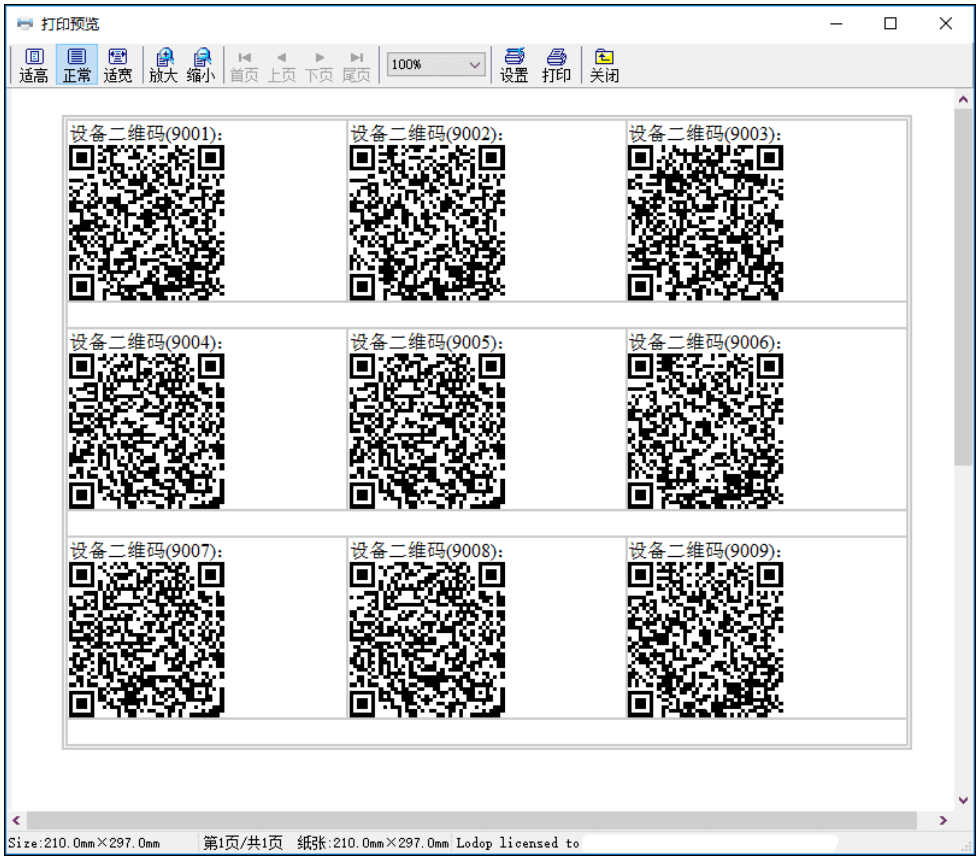
界面生成的二维码如下所示。



条码的打印效果如下所示。

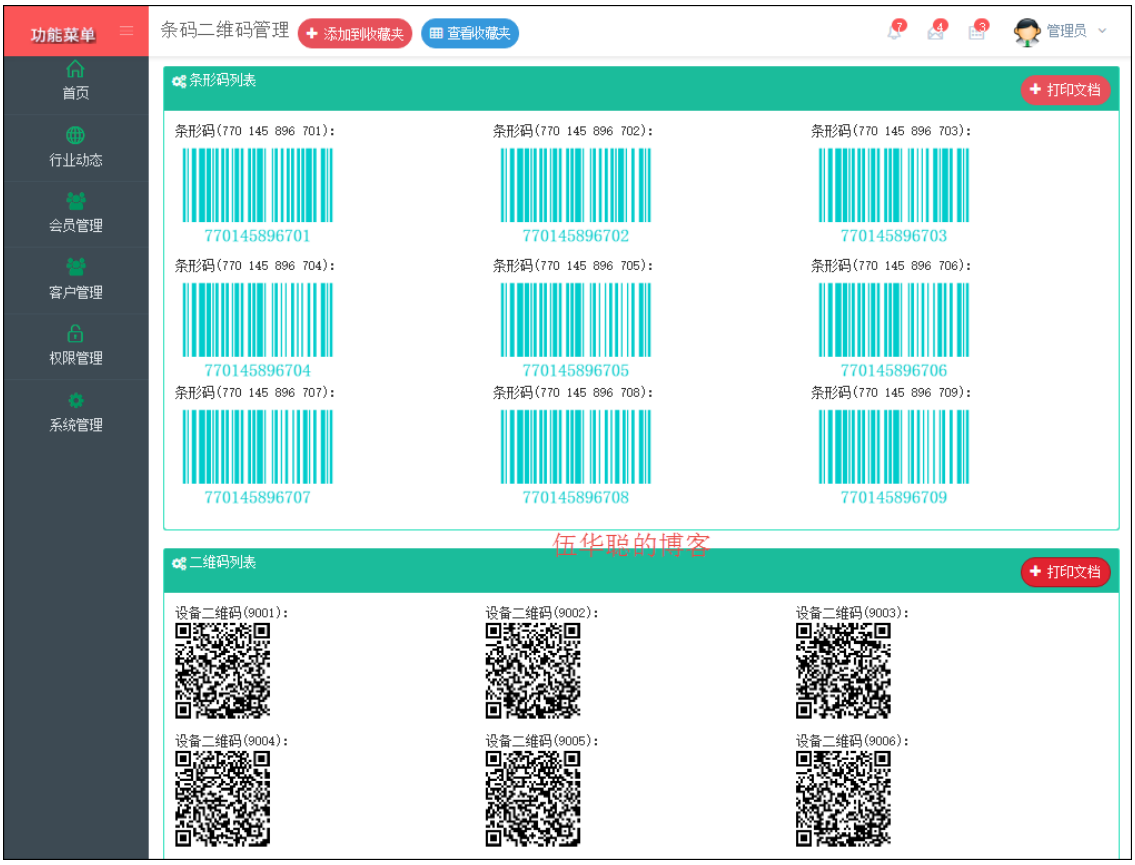


二维码打印效果如下所示。



最终完成了这个关于条形码、二维码的动态生成，以及图形打印的相关操作了。整个模

块的界面如下所示。

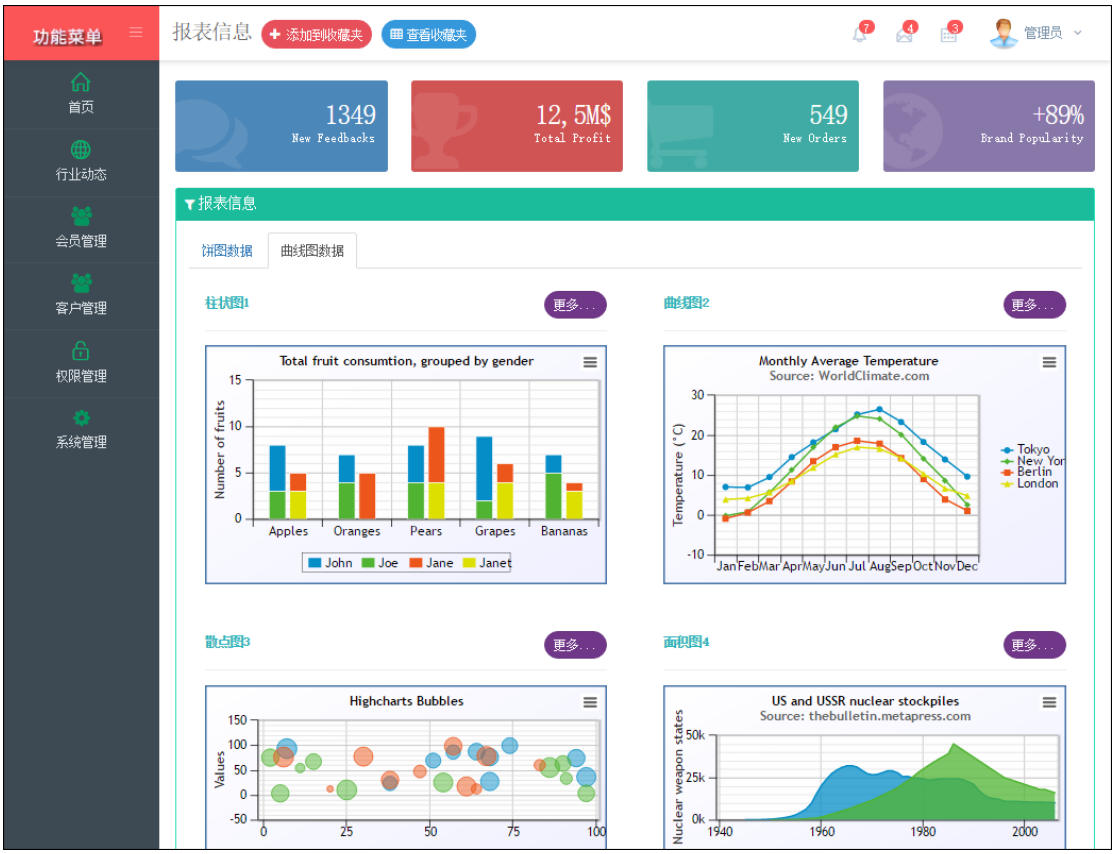


关于条形码、二维码的处理，我们这里引入的条形码组件 JsBarcode 和二维码组件 qrcodejs 是非常不错的开源 JS 组件，满足了我们大多数的要求，而且使用方便、简洁，希望这些内容能够给你的项目提供灵感及用处。

2.12. 统计图表管理

统计图表是很多应用程序需要拥有的功能，为了更好展示图表的使用操作，框架提供了多种样式的图表演示。


功能菜单
报表信息
添加到收藏夹
查看收藏夹
管理员
首页
行业动态
会员管理
客户管理
权限管理
系统管理
1349
New Feedbacks
12, 5M$
Total Profit
549
New Orders
+89%
Brand Popularity
报表信息
饼图数据
曲线图数据
图表1
更多...
集团分子公司人员组成
北京分公司: 2.4 %
上海分公司: 2.2 %
广州分公司: 95.4 %
3D图表2
集团分子公司部门组成
北京分公司
上海分公司
广州分公司
上海分公司
北京分公司
仪表图3
Speedometer
km/h
3D柱状图4
Chart rotation demo
Test options by dragging the sliders below
Values


功能菜单
报表信息
添加到收藏夹
查看收藏夹
管理员
首页
行业动态
会员管理
客户管理
权限管理
系统管理
1349
New Feedbacks
12, 5M$
Total Profit
549
New Orders
+89%
Brand Popularity
报表信息
饼图数据
曲线图数据
柱状图1
更多...
Total fruit consumtion, grouped by gender
Number of fruits
Apples
Oranges
Pears
Grapes
Bananas
John
Joe
Jane
Janet
曲线图2
Monthly Average Temperature
Source: WorldClimate.com
Temperature (°C)
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Tokyo
New Yor
Berlin
London
散点图3
Highcharts Bubbles
Values
面积图4
US and USSR nuclear stockpiles
Source: thebulletin.metapress.com
Nuclear weapon states

2.13. 在 MVC 项目中使用 RDLC 报表

RDLC 是一个不错的报表，有着比较不错的设计模式和展现效果，在我的 Winform 开发里面，使用 RDLC 也是一个比较方便操作，如可以参考文章《DevExpress 的 XtraReport 和微软 RDLC 报表的使用和对比》或者《会员管理系统的设计和开发（2）-- RDLC 报表的设计及动态加载》进行了解。但是基于 MVC 方式，一样可以构建和展现基于 RDLC 的报表。

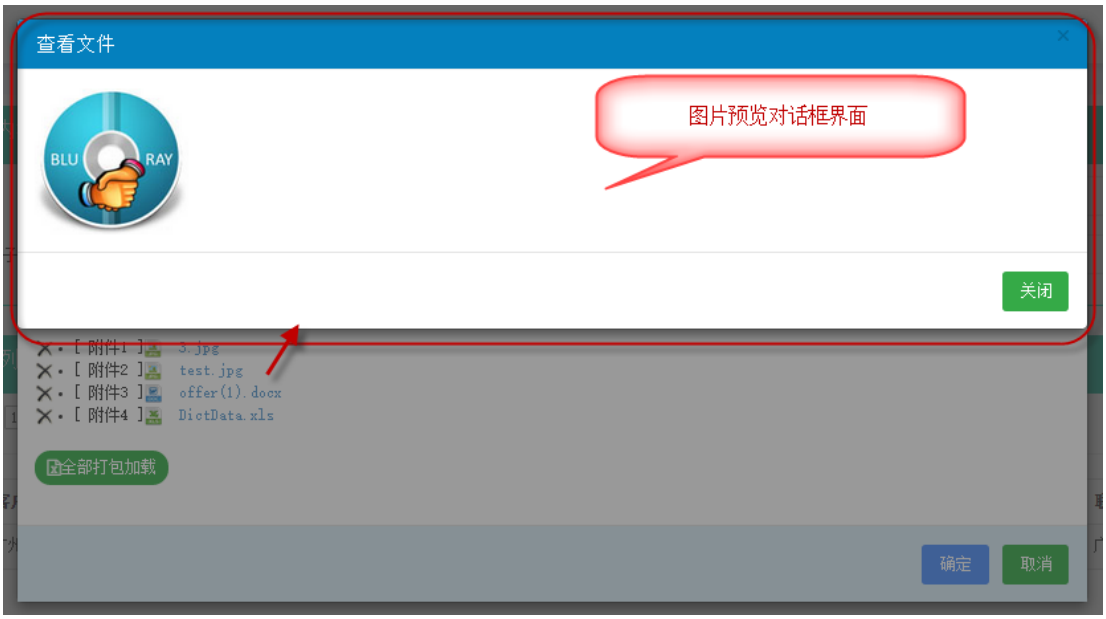
序号	用户编码	用户名/登录名	真实姓名	是否过期	移动电话	所属公司
1	shadmin	shadmin	上海管理员	正常		上海分公司
2	bjadmin	bjadmin	北京管理员	正常		北京分公司
3	gzadmin	gzadmin	广州管理员	正常		广州分公司
4	100001	100001	陈莉	正常	13810107123	广州分公司
5	100003	100003	阮琪琦	正常	13575509680	北京分公司
6	100004	100004	邓卉	正常	13917028898	广州分公司
7	100006	100006	黄晓雪	正常	18674042588	广州分公司
8	100009	100009	王景景	正常	13761601760	广州分公司
9	100012	100012	朱丽华	正常	13661246623	广州分公司
10	100015	100015	王伟	正常	13370717219	广州分公司
11	100018	100018	杨维艳	正常	13996351507	广州分公司
12	100020	100020	包英浩	正常	18628056055	广州分公司
13	100021	100021	黄志辉	正常	18696665275	广州分公司
14	100022	100022	周黎晶	正常	13452332225	广州分公司
15	100023	100023	邱海维	正常	13102321127	北京分公司
16	100024	100024	张烨	正常	18623555060	广州分公司
17	100025	100025	陈晓萌	正常	18623400401	广州分公司
18	100028	100028	黄乐瑄	正常	18602333677	广州分公司
19	100029	100029	郭锦	正常	13883844451	广州分公司
20	100030	100030	韩冬	正常	13594136753	广州分公司
21	100035	100035	徐莉	正常	18623490678	广州分公司
22	100036	100036	王海燕	正常	13824779380	广州分公司
23	100037	100037	高妮妮	正常	13616512799	广州分公司

RDLC 报表查看

2.14. 附件管理和预览

框架使用 Uploadify 文件上传组件，以及 Bootstrap 的 File Input 插件作为图片上传组件，因此能够很方便对附件进行上传和管理，并实现了对图片、Office 问答的在线预览功能，非常方便使用。

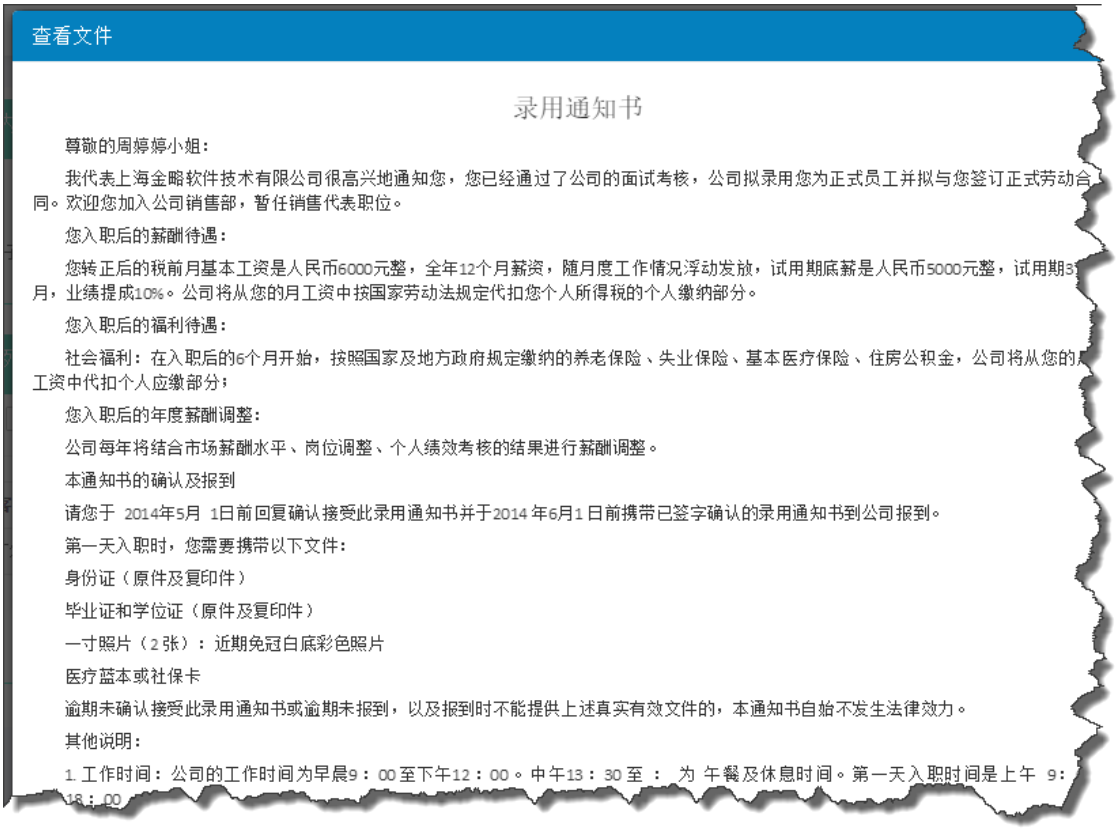
图片预览效果如下所示：



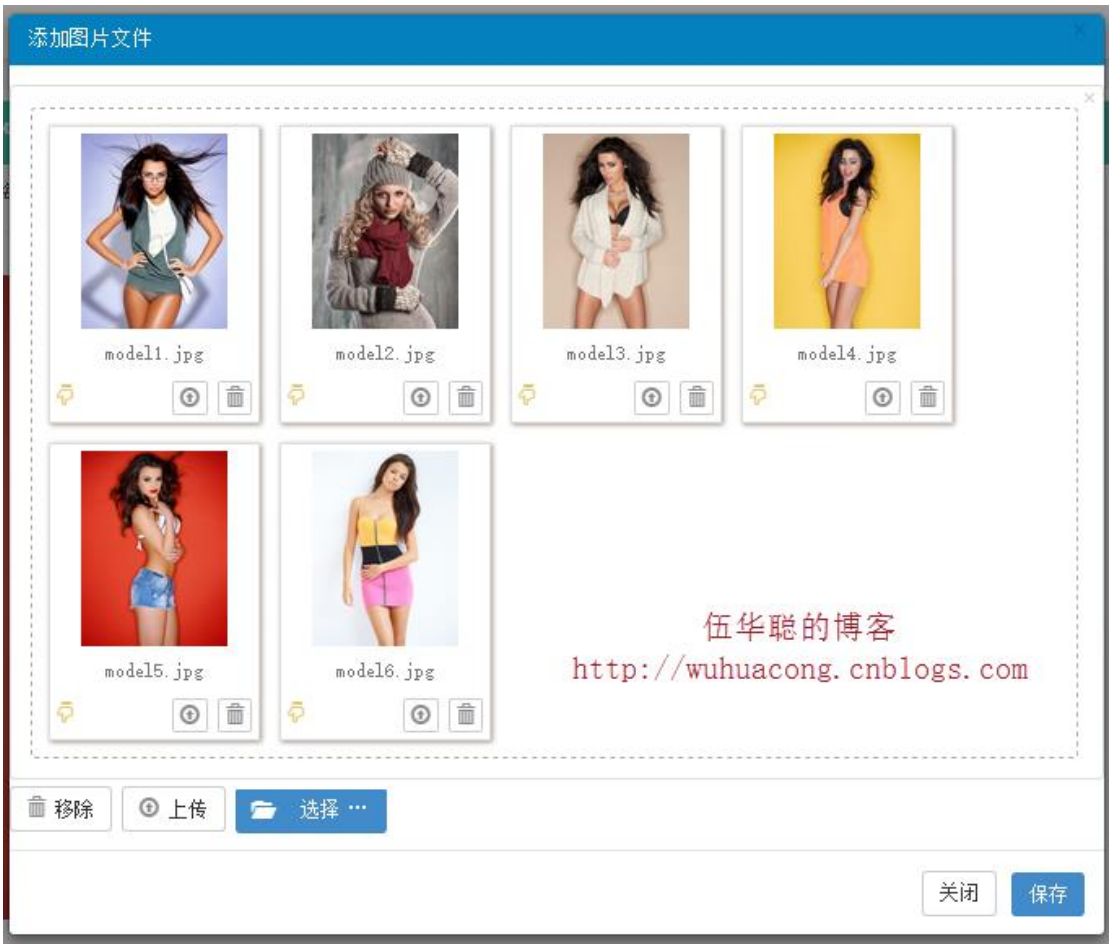
Excel 文档预览效果如下所示：

序号	用户编码	用户名/登录名	真实姓名	职务头衔	移动电话	办公电话	邮件地址	性别	QQ号码	备注
1	shadmin	shadmin	上海管理员					男	6966265	上海分公司管理员
2	bjadmin	bjadmin	北京管理员					男		北京分公司管理员
3	gzadmin	gzadmin	广州管理员					男		广州分公司管理员
4	100001	100001	陈莉		13810107123			男		
5	100003	100003	阮琪琦		13575509680			男		
6	100004	100004	邓卉		13917028898			女		
7	100006	100006	黄晓雪		18674042588			女		
8	100009	100009	王景景		13761601760			男		
9	100012	100012	朱丽华		13661246623			男		
10	100015	100015	王伟		13370717219			男		
11	100018	100018	杨维艳		13996351507			男		
12	100020	100020	包英浩		18628056055			男		
13	100021	100021	黄志辉		18696665275			女		
14	100022	100022	周黎晶		13452332225			男		
15	100023	100023	邱海维		13102321127			男		
16	100024	100024	张烨		18623555060			女		
17	100025	100025	陈晓萌		18623400401			女		
18	100028	100028	黄乐瑄		18602333677			男		
19	100029	100029	郭锦		13883844451			男		
20	100030	100030	韩冬		13594136753			男		
21	100035	100035	徐莉		18623490678			男		
22	100036	100036	王海燕		13824779380			男		
23	100037	100037	高妮妮		13616512799			男		
24	100039	100039	张乐君		13906696187			男		

WORD 文档预览效果如下所示：



我们用来构建图片相册的具体界面如下所示。



2.15. 图标管理

系统提供了一个通用的图标生成、图标选择管理模块，非常适合于动态设置菜单、按钮的图表操作，系统提供了一个对菜单图标样式动态配置的案例。

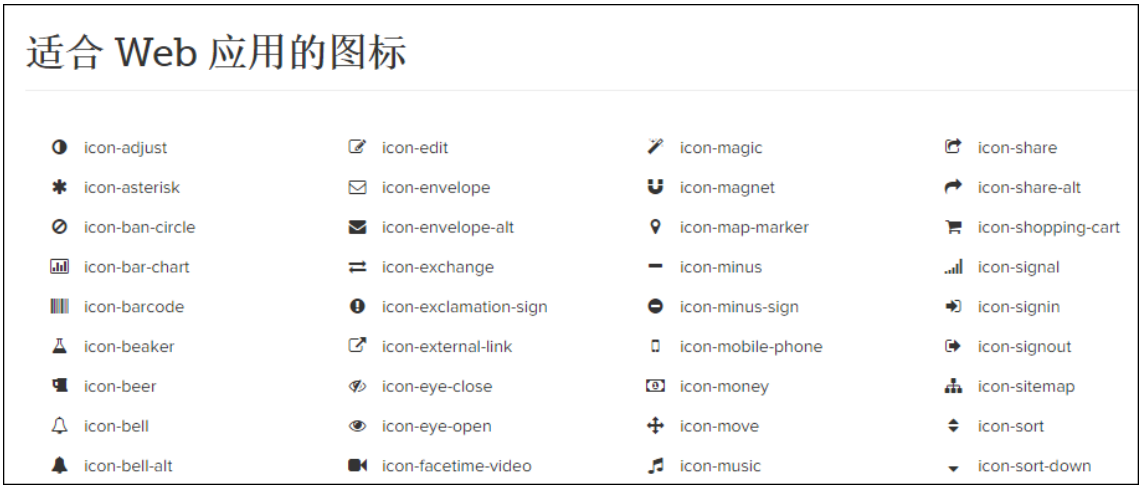
图标的展示通过 MVC 的分页处理实现，非常方便管理和查看。

2.15.1. 图标样式生成

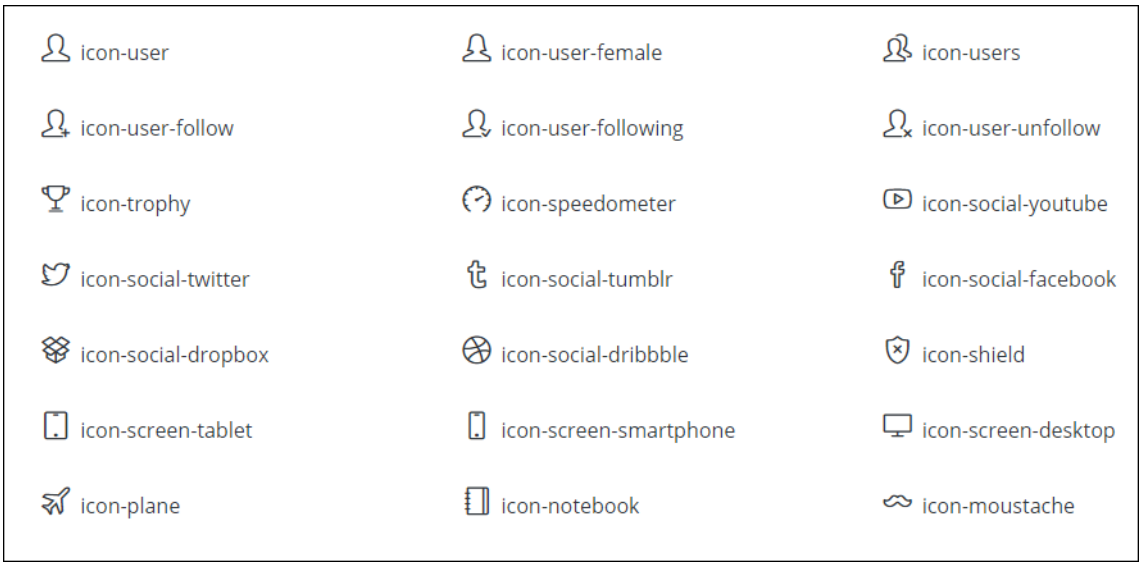
图标管理是从 Bootstrap 样式中提取出来的图表信息，存储在数据库里面供菜单等模块使用的图标数据。这些图标信息，分别从 FontAwesome、Glyphicons、SimpleLine 几个样式文件里面，通过正则表达式匹配提取，存储在数据库里面的数据记录。

Bootstrap 图标库里面分为了三类内容：

Font Awesome：Bootstrap 专用图标字体，Font Awesome 中包含的所有图标都是矢量的，也就可以任意缩放，避免了一个图标做多种尺寸的麻烦。CSS 对字体可以设置的样式也同样能够运用到这些图标上了。如下面是部分 Font Awesome 的图标：

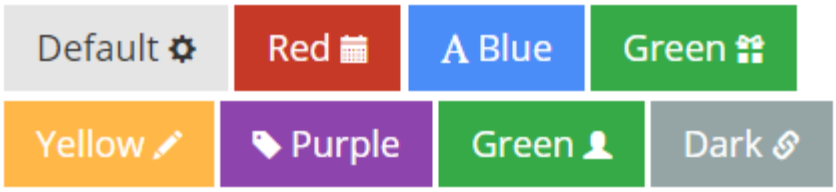


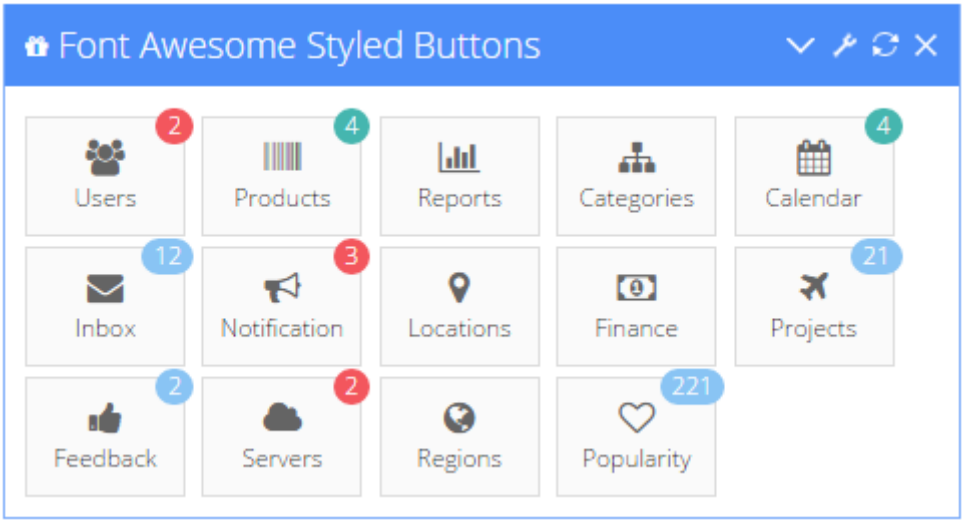
Simple Icons：收集众多网站的 Logo，并提供高质量、不同尺寸的 png 格式图片给广大网友，所有 Icon 版权归其所属公司。如下面所示是 Simple Icons 的部分图标：

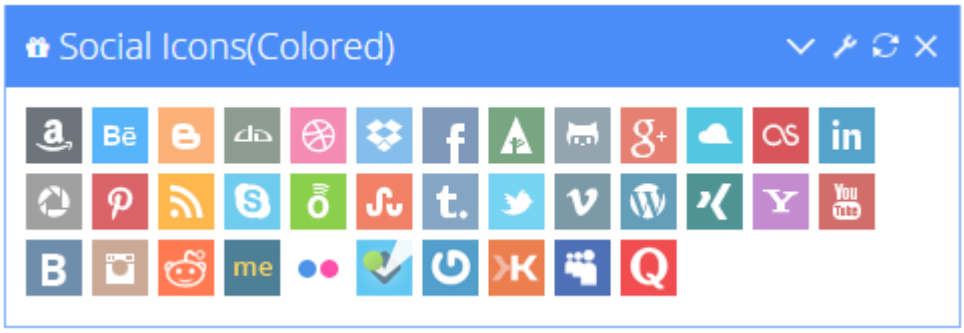


Glyphicons:包括 200 个符号字体格式图表集合，由 Glyphicons 提供，Glyphicons Halflings 一般是收费的，但是经过 Bootstrap 和 Glyphicons 作者之间的协商，允许开发人员不需要支付费用即可使用。如下是部分 Glyphicons 内容：

这几种图标，都是支持各种 Bootstrap 的主题化显示的，如下面几种效果所示。

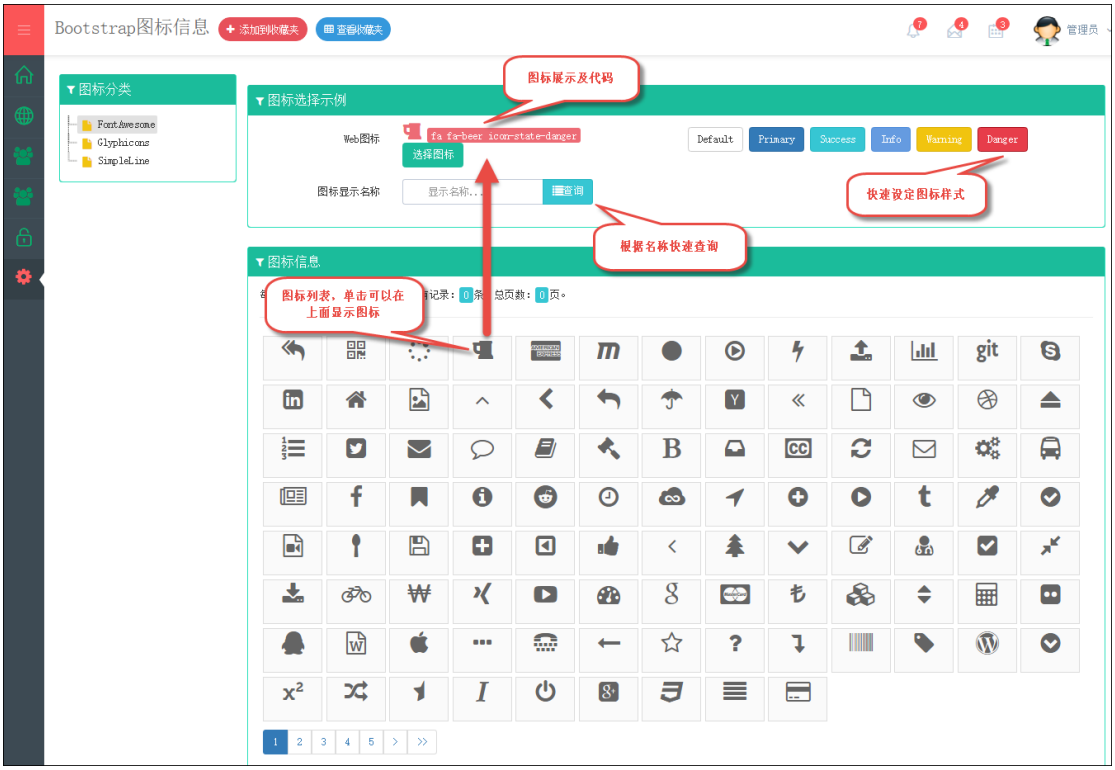




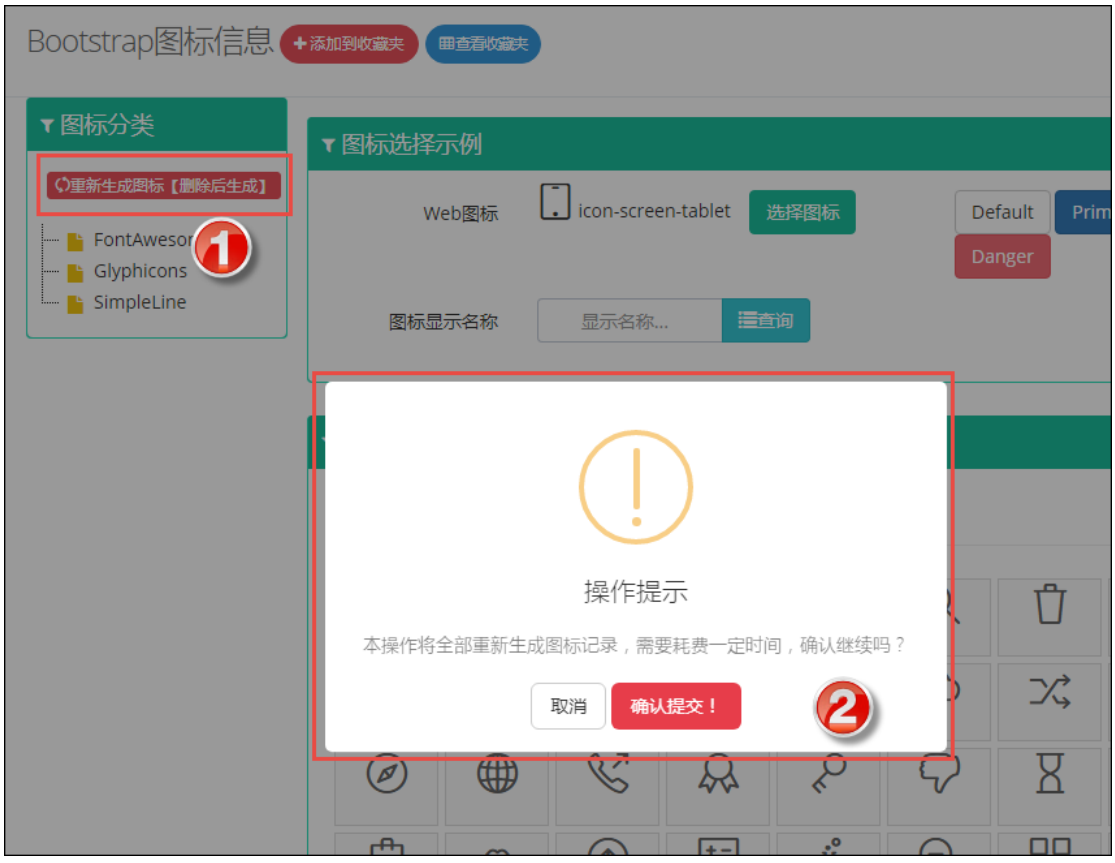


或者也可以把图标变大一些：

图标列表展示界面如下所示：



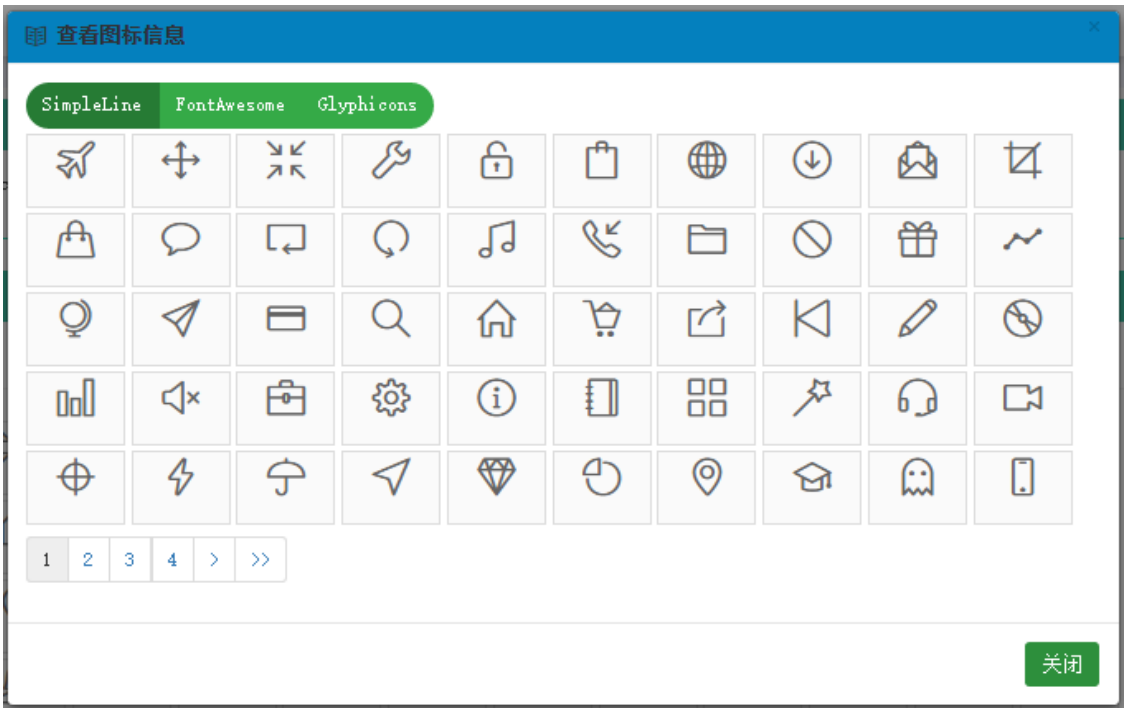
其中提取上面几种类型的图标，已经通过正则表达式进行匹配提取，封装了提取生成记录的逻辑，在界面上执行生成即可。



2.15.2. 图标选择

我们如果要能够在菜单编辑里面选择图标，那么我们还是需要把这些信息提取到数据库里面，然后展示出来给我进行选择的，否则无法做到动态配置。

当然我们选择图标的时候，提供一个弹出的对话框显示分类不同的图标，让用户选择后返回即可。



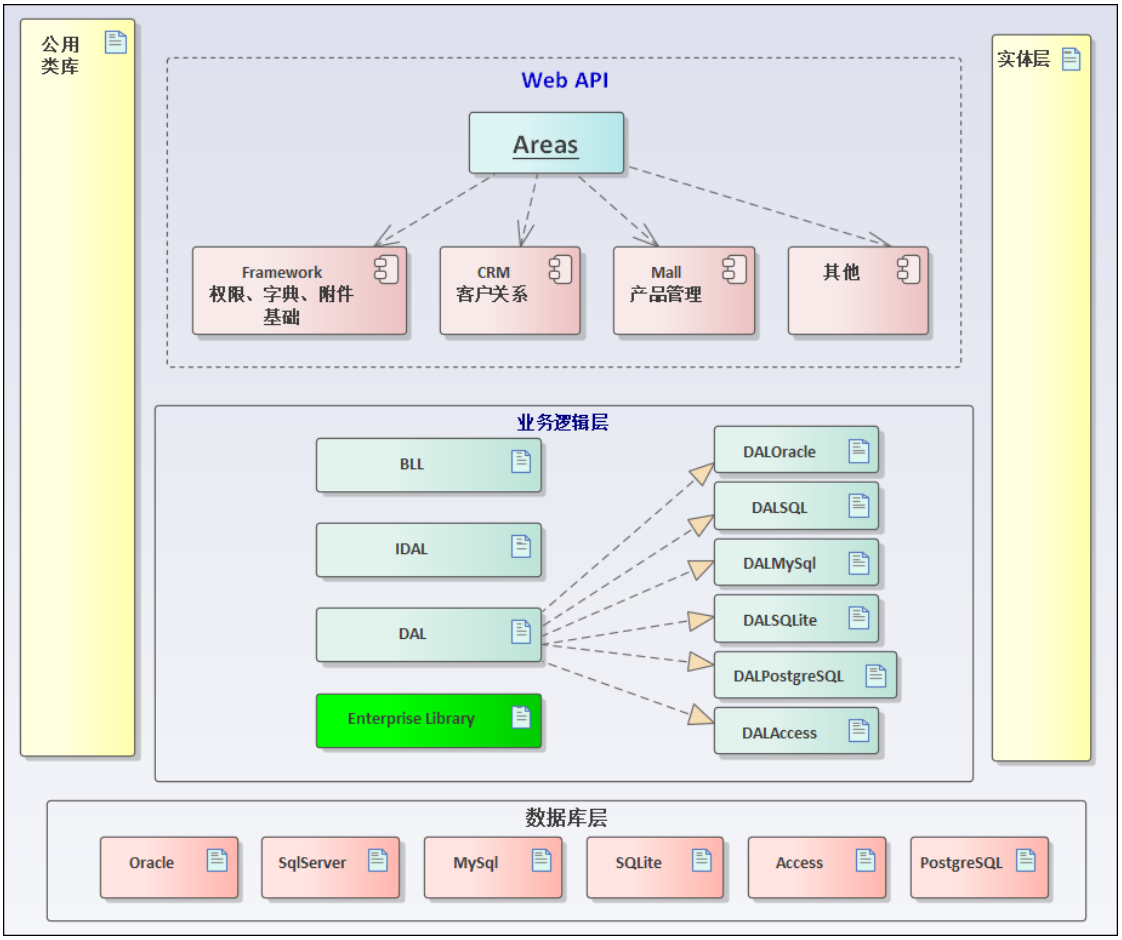
这样我们就完成了，从图标文件里面提取不同类型的图表，然后存储在数据库里面，并在页面里面显示出来，可供我们动态选择和设置了。

3. Vue&Element 框架介绍

我们在 Bootstrap 框架这个基础上扩展增加了 Vue&Element 的前端，由于 Vue&Element 的前端需要采用 Web API 的接口，我们在这个基础上借鉴了 ABP 框架的 Web API 接口处理及前端管理界面的内容，在 Bootstrap 开发框架基础上增加 WebApi+Vue&Element 的前端。

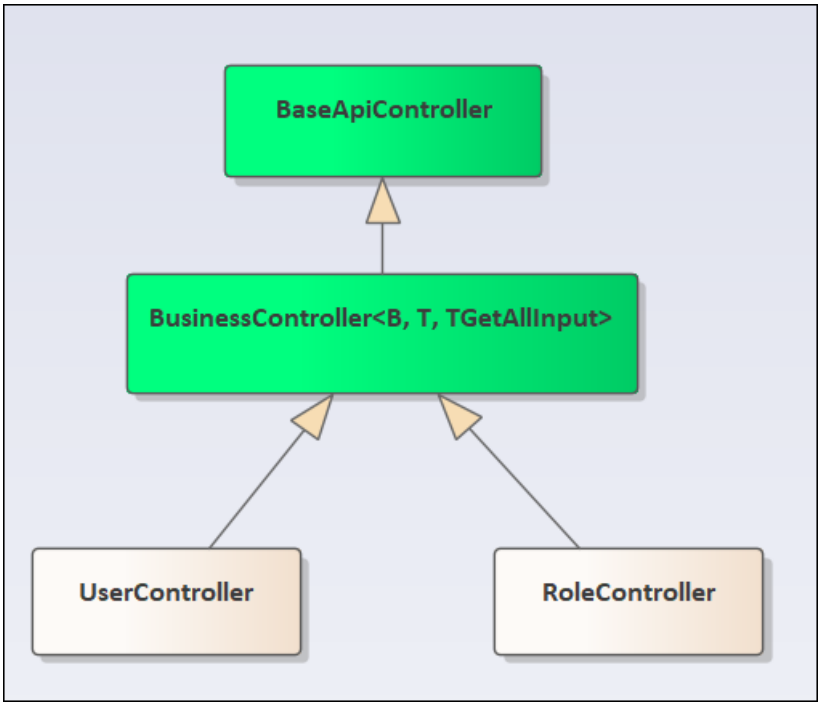
3.1. Web API 后端的架构设计

这个是属于前端、后端完全分离的架构设计，后端采用基于 Asp.net 的 Web API 技术，并提供按域来管理 API 的分类，Web API 如下架构所示。



通过上面的架构设计，可以看出，底层 BLL、DAL、Entity、IDAL、公用类库等分层都是完全通用的，不同的是，我们后端只是提供 Web API 的接口服务给前端，这个和我们的《ABP 框架使用》的理念一致，前后端分离。

为了更好的进行后端 Web API 控制器的相关方法的封装处理，我们把一些常规的接口处理放在 BaseApiController 里面，而把基于业务表的操作接口放在 BusinessController 里面定义，如下所示。



在 BaseApiController 里面，我们使用了结果封装和异常处理的过滤器统一处理。

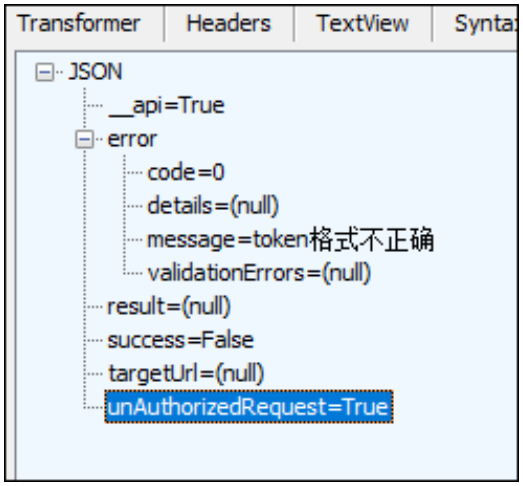
```
/// <summary>
/// 所有接口基类
/// </summary>
[ExceptionHandling]
[WrapResult]
public class BaseApiController : ApiController
```

其中 ExceptionHandling 和 WrapResult 的过滤器处理，可以参考我的随笔《利用过滤器 Filter 和特性 Attribute 实现对 Web API 返回结果的封装和统一异常处理》进行详细了解。

而业务类的接口通用封装，则放在了 BusinessController 控制器里面，其中使用了泛型定义，包括实体类，业务操作类，分页条件类等内容作为约束参数，如下所示。

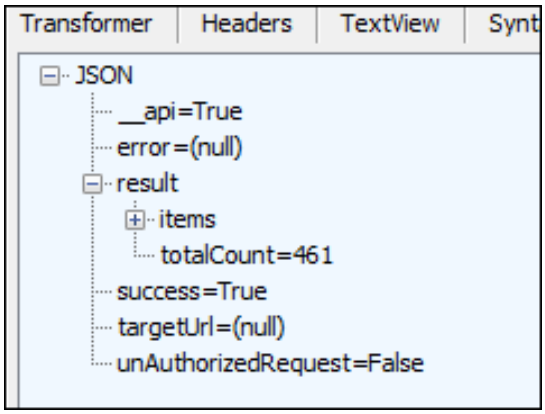
```
/// <summary>
/// 本控制器基类专门为访问数据业务对象而设的基类
/// </summary>
/// <typeparam name="B">业务对象类型</typeparam>
/// <typeparam name="T">实体类类型</typeparam>
[ApiAuthorize]
public class BusinessController<B, T, TGetAllInput> : BaseApiController
    where B : class
    where TGetAllInput : IPagedAndSortedResultRequest
    where T : BaseEntity, new()
```

通过 Web API 接口返回结果的统一封装处理，我们定义了相关的格式如下所示。



其中 result 是返回对应的对象信息，如果没有则返回 null.

如果是分页查询返回结果集合，其结果如下所示。



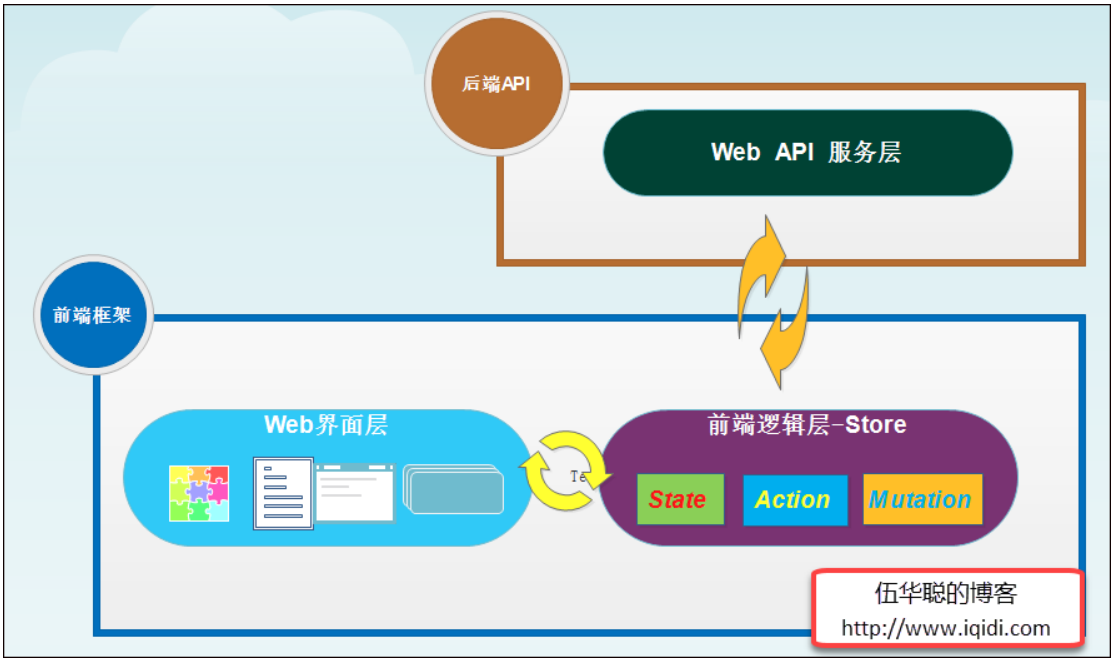
展开单条记录明细如下所示。

```
"result": {
  "totalCount": 461,
  "items": [
    {
      "pid": -1,
      "nickname": "",
      "isExpire": false,
      "expireDate": null,
      "title": "",
      "identityCard": "",
      "officePhone": "",
      "homePhone": "",
      "address": "",
      "workAddr": "",
      "gender": "男",
      "birthday": "1900-01-01 00:00:00",
      "qq": "",
      "signature": "",
      "auditStatus": "",
      "portrait": null,
      "note": "北京分公司管理员",
      "customField": "",
      "deptName": "财务部",
      "companyName": "北京分公司",
      "sortCode": "",
      "creator": "管理员",
      "creator_ID": "1",
      "createTime": "2013-12-31 10:20:36",
      "editor": "管理员",
      "editor_ID": "1",
      "editTime": "2014-11-06 14:42:36",
      "deleted": false,
      "currentLoginIP": "",
      "currentLoginTime": "1900-01-01 00:00:00",
      "currentMacAddress": "",
      "openId": "",
      "status": "",
      "subscribeWechat": "",
      "deptPermission": "",
      "corpUserId": "",
      "corpStatus": "",
      "roleNames": null,
      "roles": null,
      "isSuperAdmin": false,
      "isAdmin": false,
      "ouNames": null,
      "id": 403,
```


3.2. Vue&Element 的前端的架构设计

而 Vue&Element 的前端的架构设计，也借鉴了我们 ABP 框架的前端管理部分，详细也可以了解下我的相关随笔《循序渐进 VUE+Element》。

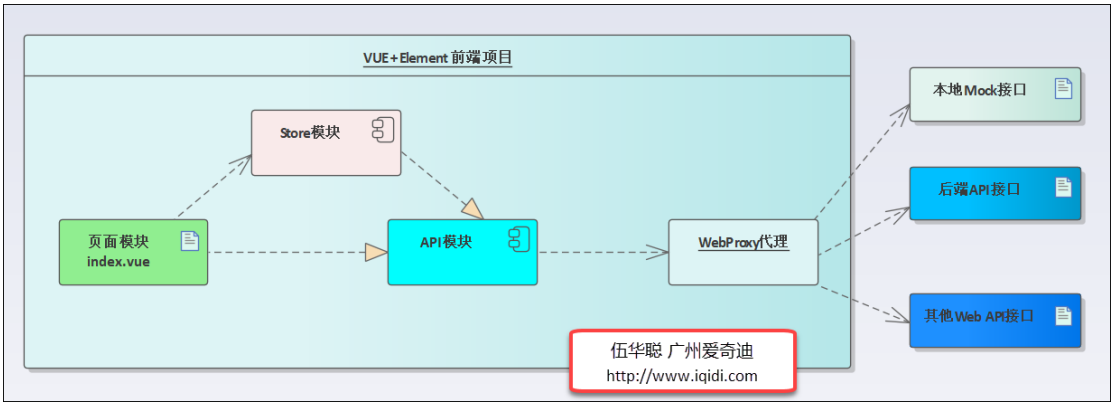
Vue&Element 的前端的架构设计如下所示。



引入了前后端分离的 Vue + Element 作为前端技术路线，那么前后端的边界则非常清晰，前端可以在通过网络获取对应的 JSON 就可以构建前端的应用了。

一般来说，我们页面模块可能会涉及到 Store 模块，用来存储对应的状态信息，也可能是直接访问 API 模块，实现数据的调用并展示。在页面开发过程中，多数情况下，不需要 Store 模块进行交互，一般只需要存储对应页面数据为全局数据状态的情况下，才可能启用 Store 模块的处理。

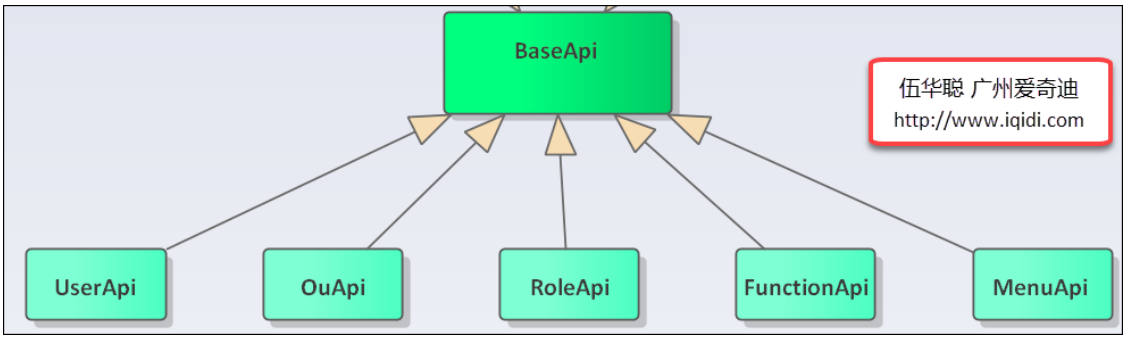
通过 WebProxy 代理的处理，我们可以很容易在前端中实现跨域的处理，不同的路径调用不同的域名地址 API 都可以，最终转换为本地的 API 调用，这就是跨域的处理操作。



前端根据 ABP 后端的接口进行前端 JS 端的类的封装处理，引入了 ES6 类的概念实现业务基类接口的统一封装，简化代码。

权限模块我们涉及到的用户管理、机构管理、角色管理、菜单管理、功能管理、操作日

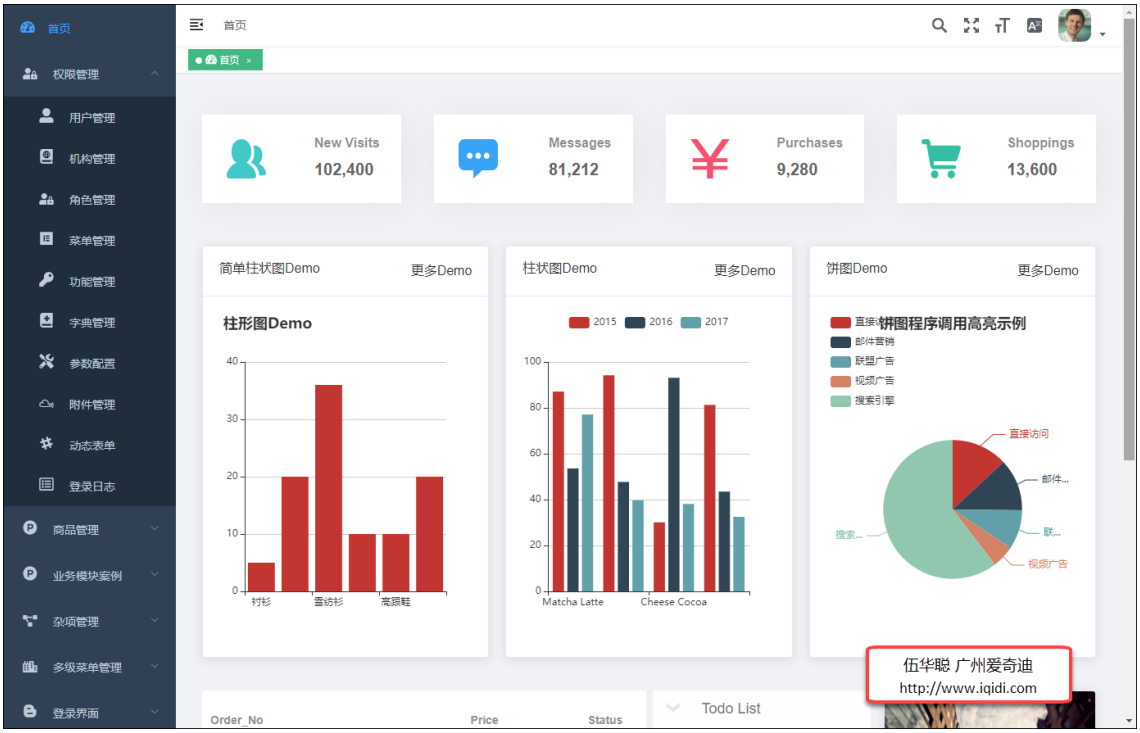
志、登录日志等业务类，那么这些类继承 BaseApi，就会具有相关的接口了，如下所示继承关系。



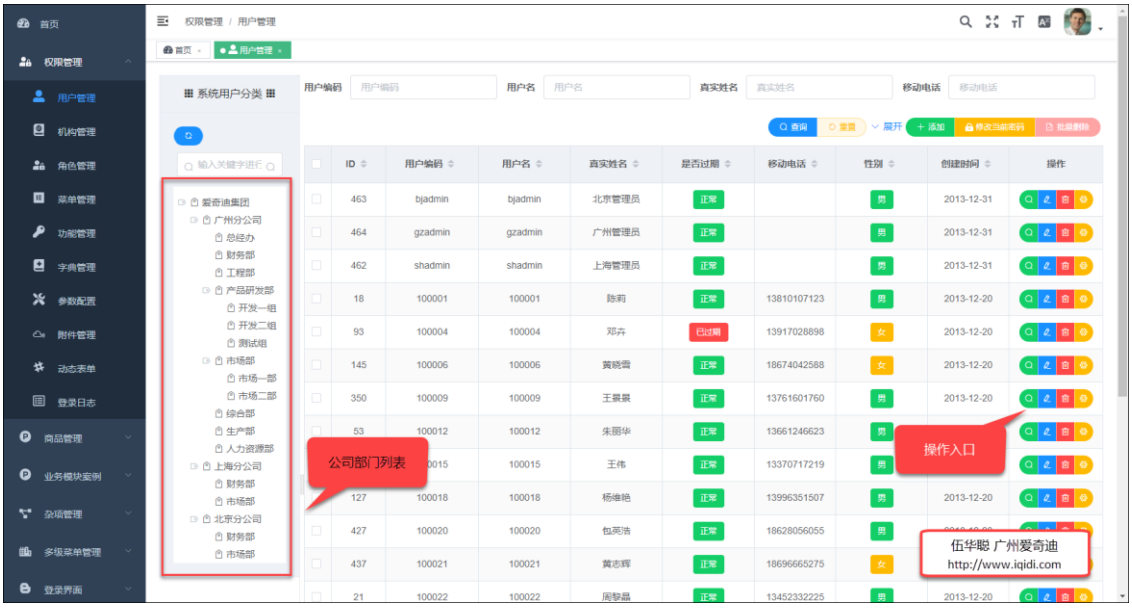
3.3. WebApi+Vue&Element 的前端界面展示

主体框架界面采用的是基于后台配置的菜单动态生成，左侧是菜单，右边顶部是特定导航条和内容区，这个和我们 ABP 框架的前端界面是一致的。

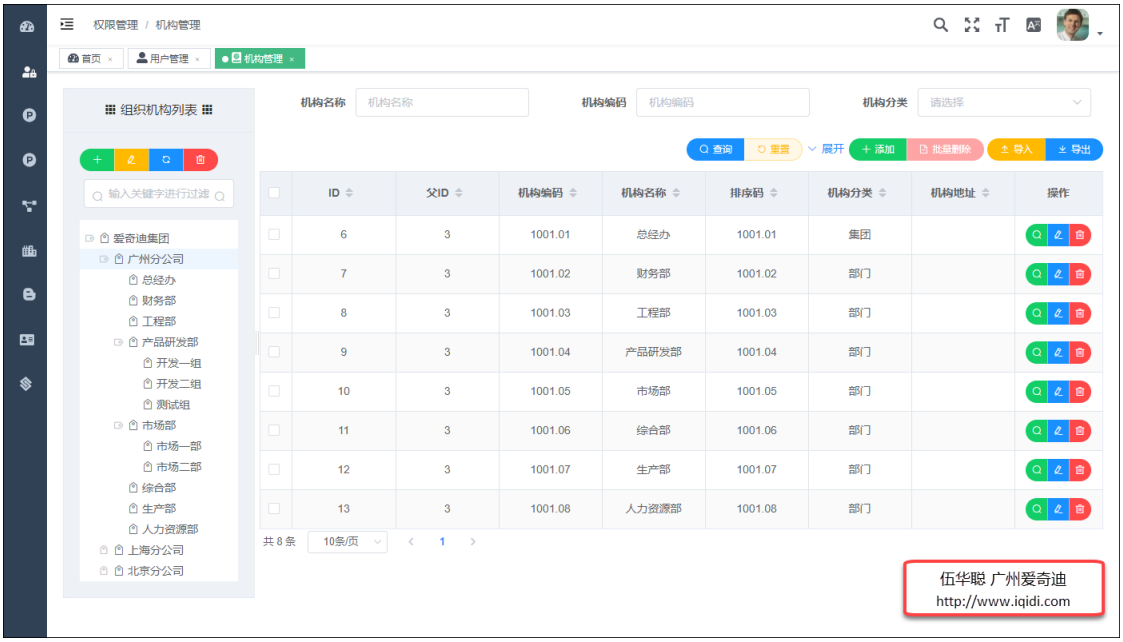
系统主界面的开发，基本上都是标准的界面，在系统左侧放置系统菜单，右侧中间区域则放置列表展示内容，但是在系统菜单比较多的时候，就需要把菜单分为几级处理。系统菜单在左侧放置一个自定义菜单组件列表，这样通过树形列表的收缩折叠，就可以放置非常多的菜单功能了。



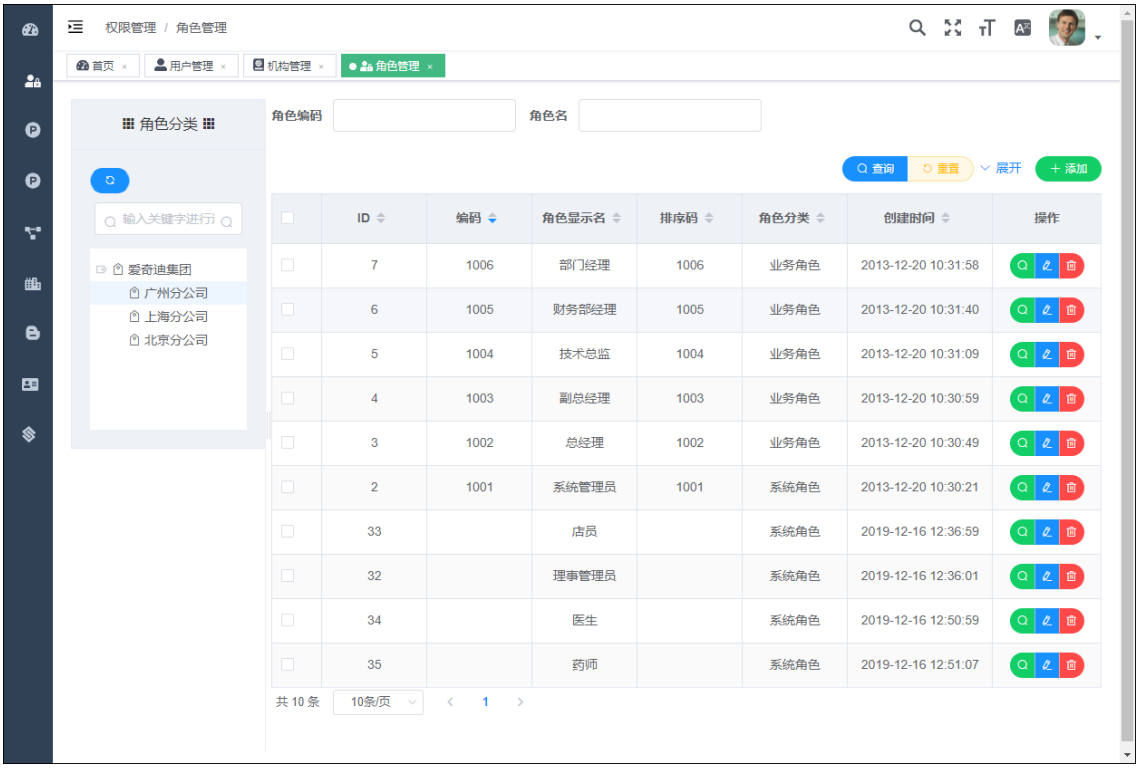
用户管理界面，沿袭 Bootstrap 框架的布局进行管理，通过用户机构方式，快速展示用户分页列表，如下界面所示。



机构管理界面如下所示。



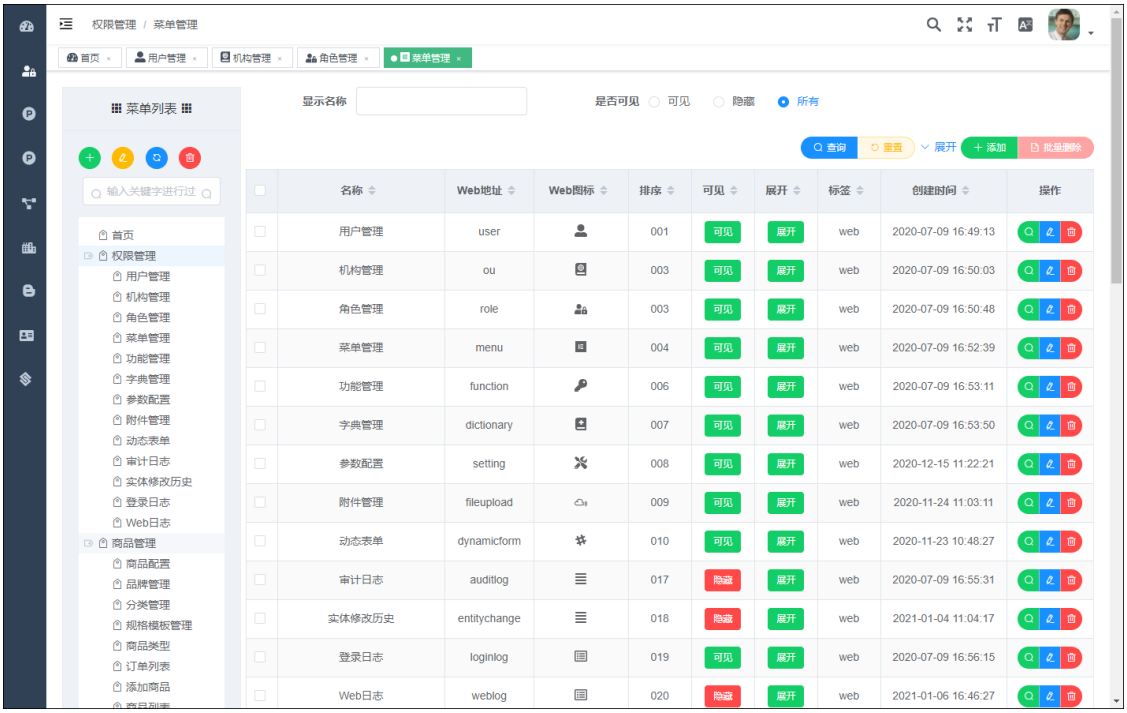
角色管理界面如下所示。



其角色的编辑界面如下所示，包括了基础信息、用户、菜单、权限等项目。

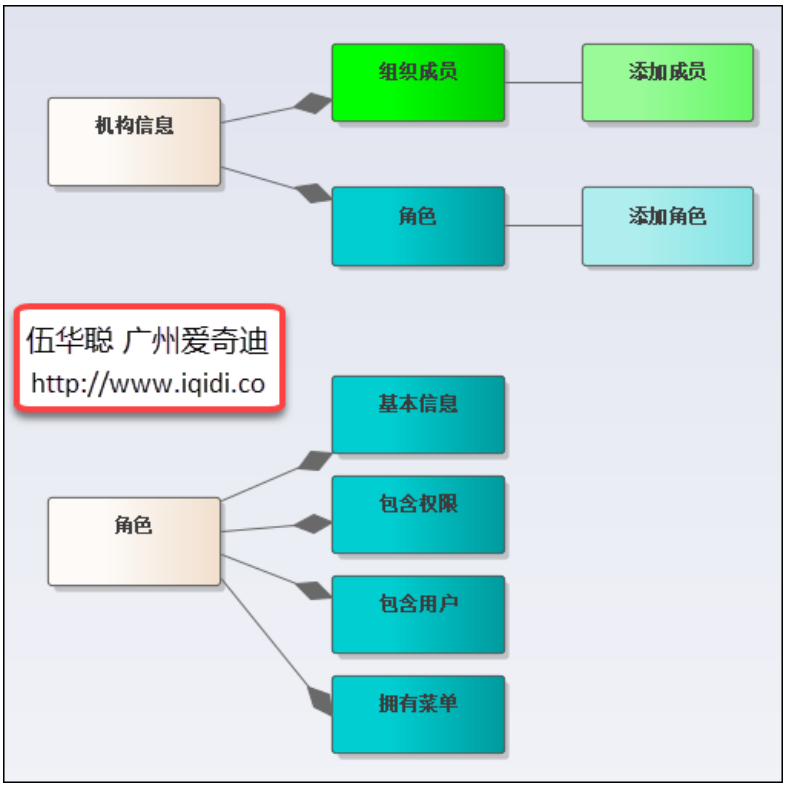
整个系统的菜单，既可以通过默认 Mock 的初始菜单，也可以通过后端 API 获得的菜单资源，动态在界面上进行展示，系统界面左侧的菜单是动态获取并展示出来的，结合路由的判断可以限制用户访问的菜单权限。

菜单管理界面如下所示。



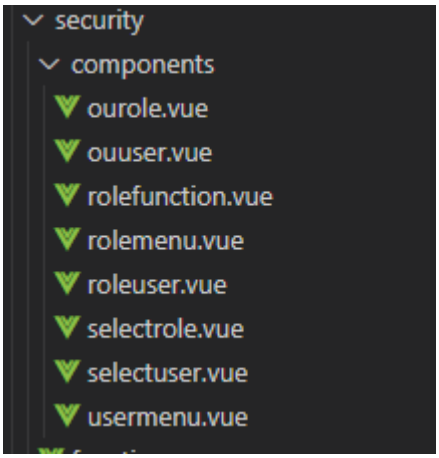
菜单资源在角色管理中分配给具体角色，即可实现对菜单的动态控制管理了。

前端的界面管理，依旧可以通过分拆模块的方式，完成更加方便的组合处理



利用模块化的处理方式，我们可以把界面部分内容作为一个组件进行封装处理，如在

权限管理中，我们定义了一部分重用的组件界面，如下所示。



定义好各种界面的自定义组件后，在主界面中进行 Import 导入使用即可，非常方便重用。

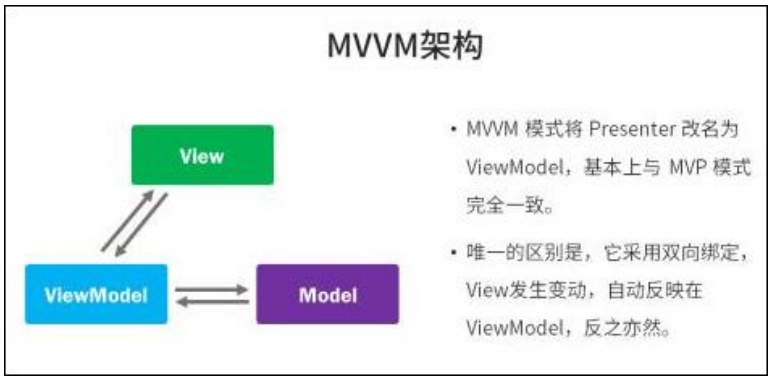
```
// 业务API对象
import ou from '@/api/system/ou'
import role from '@/api/system/role'

// 角色包含用户界面组件
import roleuser from './components/roleuser'
// 角色拥有菜单界面组件
import rolemenu from './components/rolemenu'
// 角色包含权限界面组件
import rolefunction from './components/rolefunction'
```

3.4. Vue&Element 基础介绍

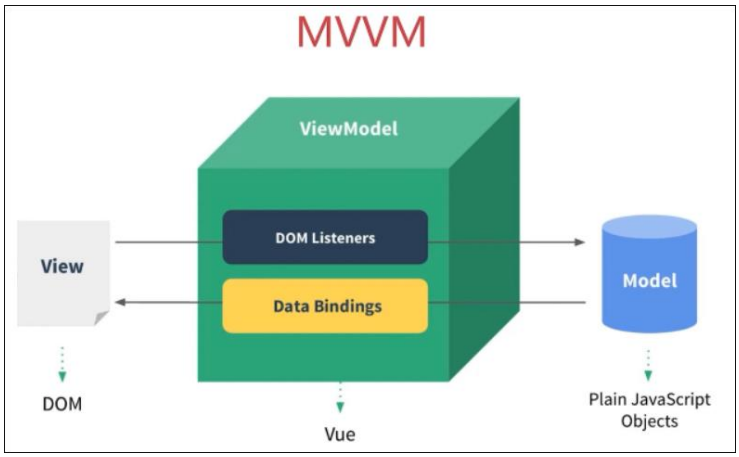
3.4.1. Vue 框架简介

Vue 是一套构建用户界面的框架， 开发只需要关注视图层， 它不仅易于上手，还便于与第三方库或既有项目的整合；另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。是基于 MVVM（Model-View-ViewModel）设计思想。提供 MVVM 数据双向绑定的库，专注于 UI 层面。



View 就是 DOM 层，ViewModel 就是通过 new Vue()的实例对象，Model 是原生 js。开发者修改了 DOM，ViewModel 对修改的行为进行监听，监听到了后去更改 Model 层的数据，然后再通过 ViewModel 去改变 View，从而达到自动同步。

Vue 核心思想，包括数据驱动、组件化等方面。



1、数据驱动

数据驱动（数据双向绑定），　在 Vue 中，Directives 对 view 进行了封装，当 model 中的数据发生变化时，Vue 就会通过 Directives 指令去修改 DOM，同时也通过 DOM Listener 实现对视图 view 的监听，当 DOM 改变时，就会被监听到，实现 model 的改变，从而实现数据的双向绑定。

2、组件化

组件化就是实现了扩展 HTML 元素，封装可用的代码。

1）页面上每个独立的可视/可交互区域视为一个组件。

2）页面不过是组件的容器，组件可以嵌套自由组合形成完整的页面

3.4.2. Element 介绍

Element，是饿了么公司开发的一套 UI 组件，一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端组件库。提供了配套设计资源，帮助你的开发快速成型。由饿了么公司前端团队开源。

Element 前端界面套件，提供了几乎所有常见的 Web 组件封装，并扩展了很多功能丰富的 UI 组件，使用这些界面组件可以极大提高 Web 界面的开发效率，同时也能够把这些基础组件封装层更高级、功能更丰富的自定义组件。

3.4.3. VS code 的安装

有别于之前的 Asp.net 的开发，纯前端的开发，一般不会再采用笨重的 VS 进行前端的开发，而改用 VS Code 或者 WebStorm 等轻型的开发工具来进行前端代码的开发和维护，虽然是轻型开发工具，不过功能也是非常强大的，而且开发环境可以在 Windows 系统，也可以在 Mac 系统等，实现多平台的开发环境。

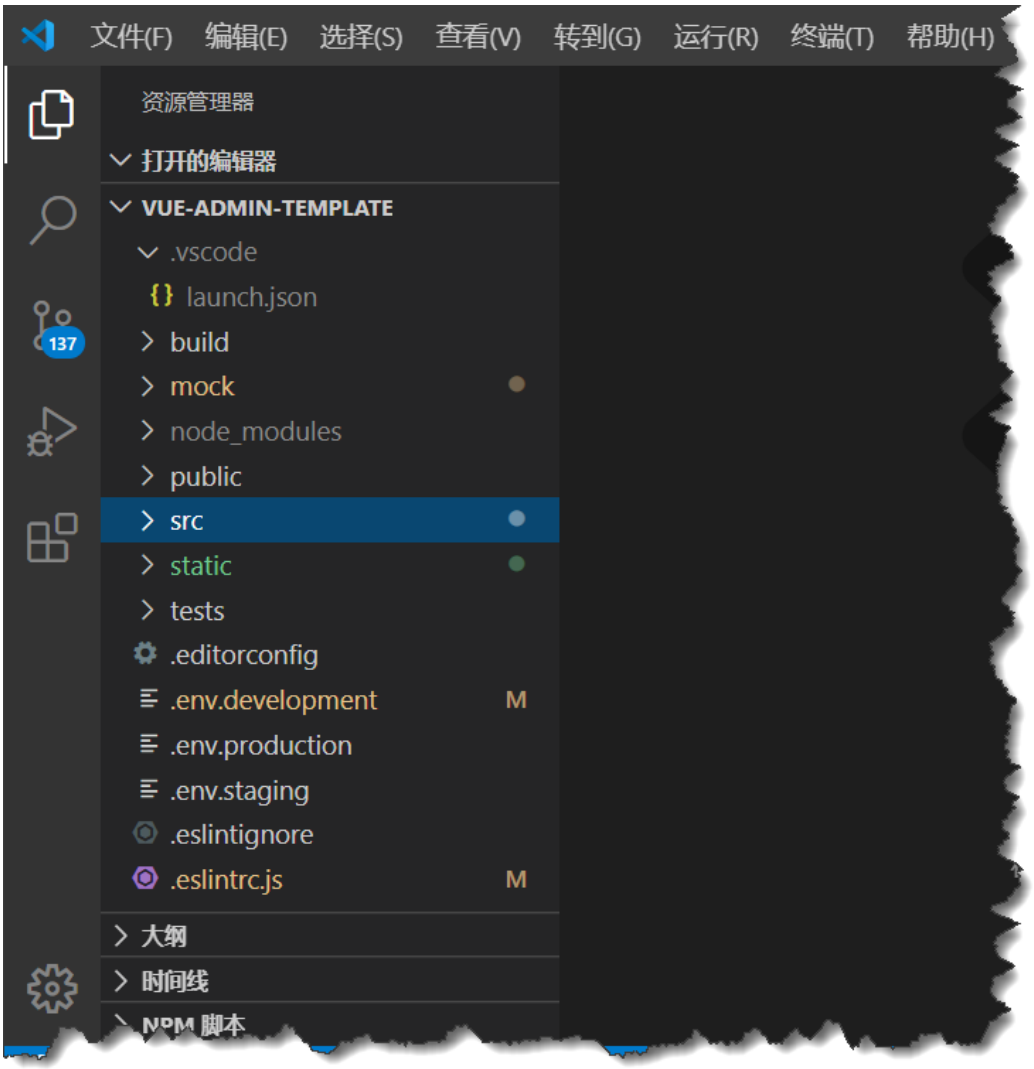
VS Code（Visual Studio Code）是由微软研发的一款免费、开源的跨平台文本（代码）编辑器。几乎完美的编辑器。

官网：https://code.visualstudio.com

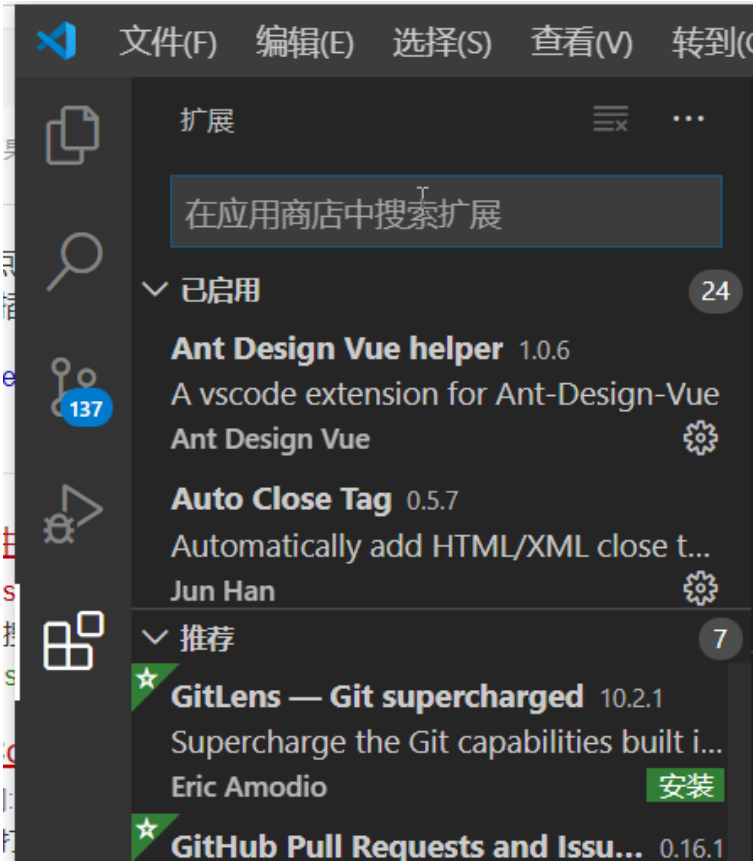
文档：https://code.visualstudio.com/docs

源码：https://github.com/Microsoft/vscode

VS Code 的界面大概如下所示，一般安装后，如果为英文界面，则安装它的中文包即可。



VS Code 安装后，我们一般还需要搜索安装一些所需要的插件辅助开发。安装插件很简单，在搜索面板中查找到后，直接安装即可。



一般我们需要安装这些 vs code 插件:

Vetur

Vue 多功能集成插件，包括：语法高亮，智能提示，emmet，错误提示，格式化，自动补全，debugger。vscode 官方钦定 Vue 插件，Vue 开发者必备。

ESLint

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。

而 VSCode 中的 ESLint 插件就直接将 ESLint 的功能集成好，安装后即可使用，对于代码格式与规范的细节还可以自定义，并且一个团队可以共享同一个配置文件，这样一个团队所有人写出的代码就可以使用同一个代码规范，在代码签入前每个人可以完成自己的代码规范检查。

VS Code - Debugger for Chrome 结合 Chrome 进行调试的插件

此工具简直就是前端开发必备，将大大地改变你的开发与调试模式。

以往的前端调试，主要是 JavaScript 调试，你需要在 Chrome 的控制台中找到对应代码的部分，添加上断点，然后在 Chrome 的控制台中单步调试并在其中查看值的变化。

而在使用了 Debugger for Chrome 后，当代码在 Chrome 中运行后，你可以直接在 VSCode 中加上断点，点击运行后，Chrome 中的页面继续运行，执行到你在 VSCode 中添加的断点后，你可以直接在 VSCode 中进行单步调试。

Beautify

Beautify 插件可以快速格式化你的代码格式，让你在编写代码时杂乱的代码结构瞬间变得非常规整，代码强迫症必备，较好的代码格式在后期维护以及他人阅读时都会有很多的便利。

3.4.4. 安装 node 开发环境

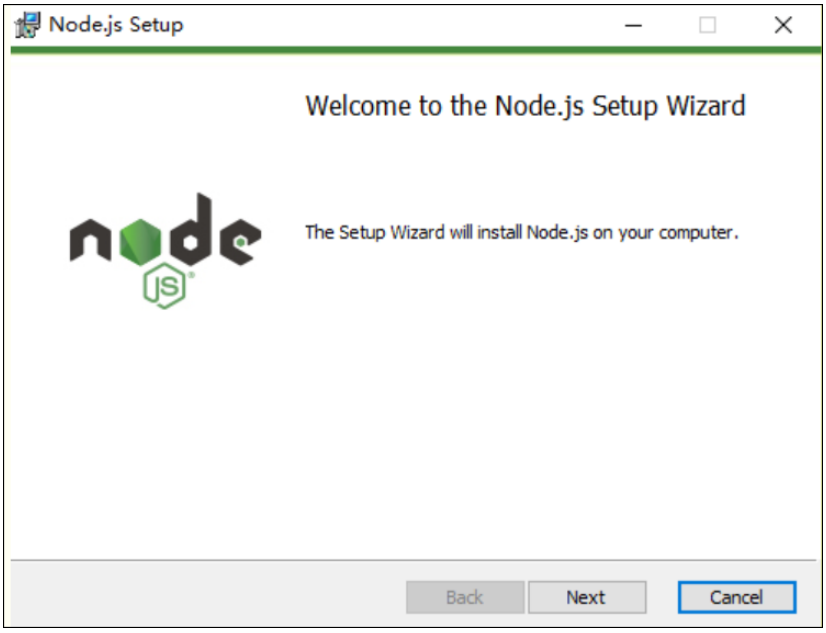
利用 VS Code 开发，我们很多时候，需要使用命令行 npm 进行相关模块的安装，这些需要 node 环境的支持，安装好 node 后，npm 也就一起安装好了。

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。

Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。

Node.js 的包管理器 npm，是全球最大的开源库生态系统。

node 下载：https://nodejs.org/en/



安装后，我们可以通过命令行或者 VS Code 里面的 Shell 进行查看 node 和 npm 的版本号了

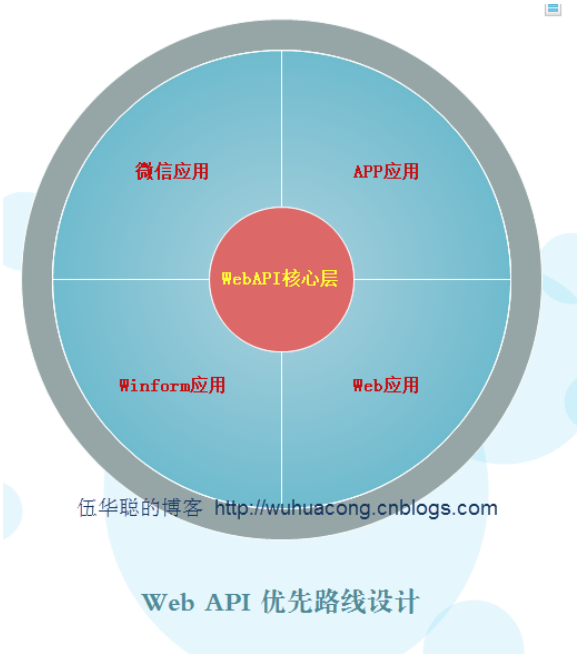
node -v

npm -v

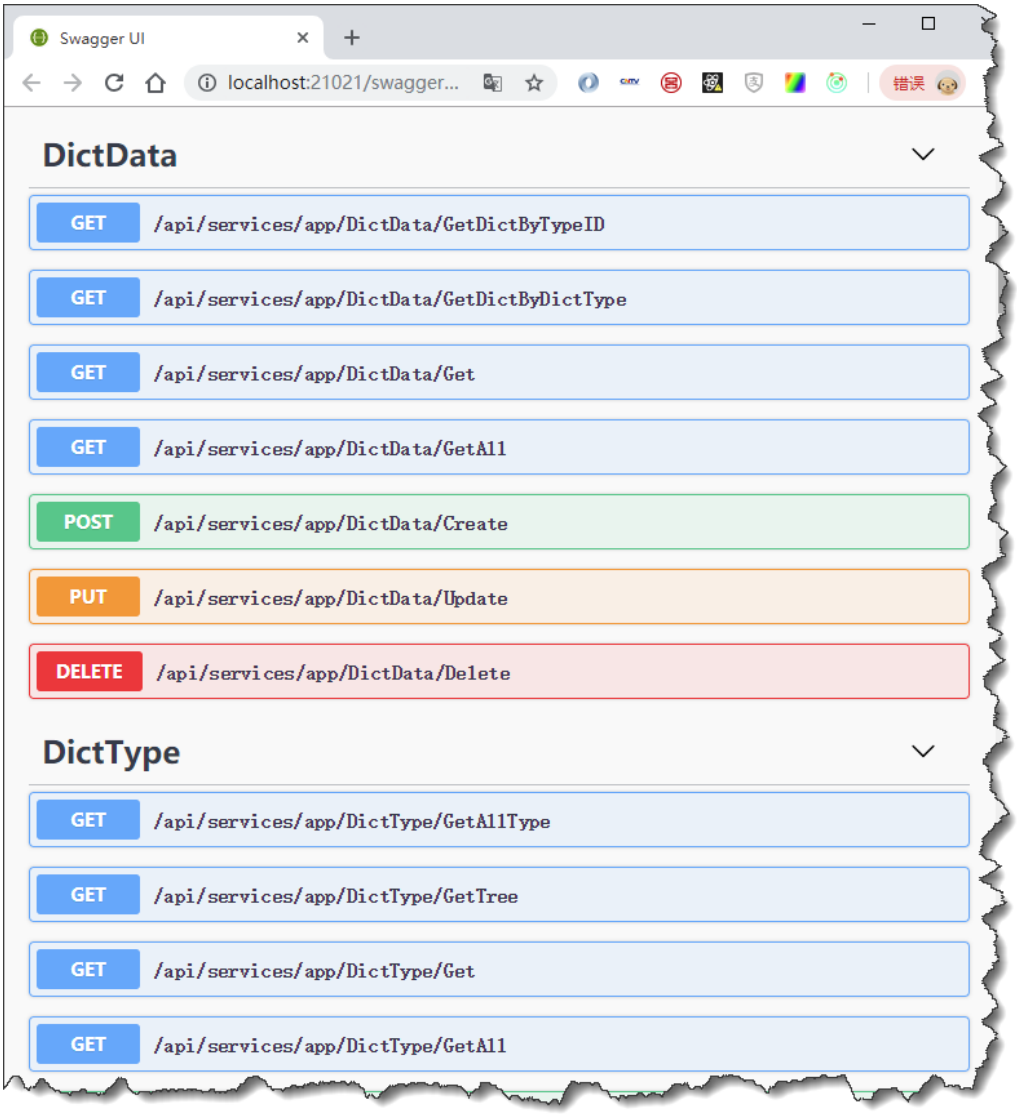
3.4.5. 前后端的分离和 Web API 优先路线设计

Web API 是一种应用接口框架，它能够构建 HTTP 服务以支撑更广泛的客户端（包括浏览器，手机和平板电脑等移动设备）的框架， ASP.NET Web API 是一种用于在 .NET Framework/ .net Core 上构建 RESTful 应用程序的理想平台。

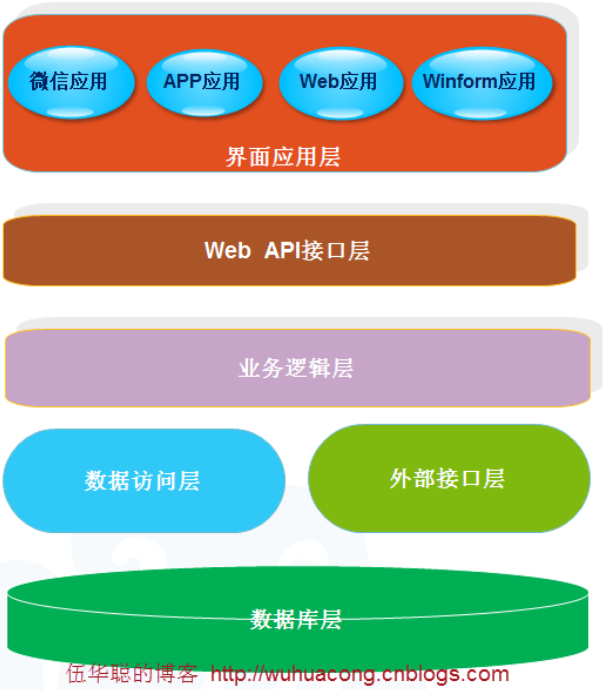
在目前发达的应用场景下，我们往往需要接入 Winform 客户端、APP 程序、网站程序、以及微信应用等程序，这些数据应该可以由同一个服务提供，这个就是我们所需要构建的 Web API 平台。由于 Web API 层作为一个公共的接口层，我们就很好保证了各个界面应用层的数据一致性。



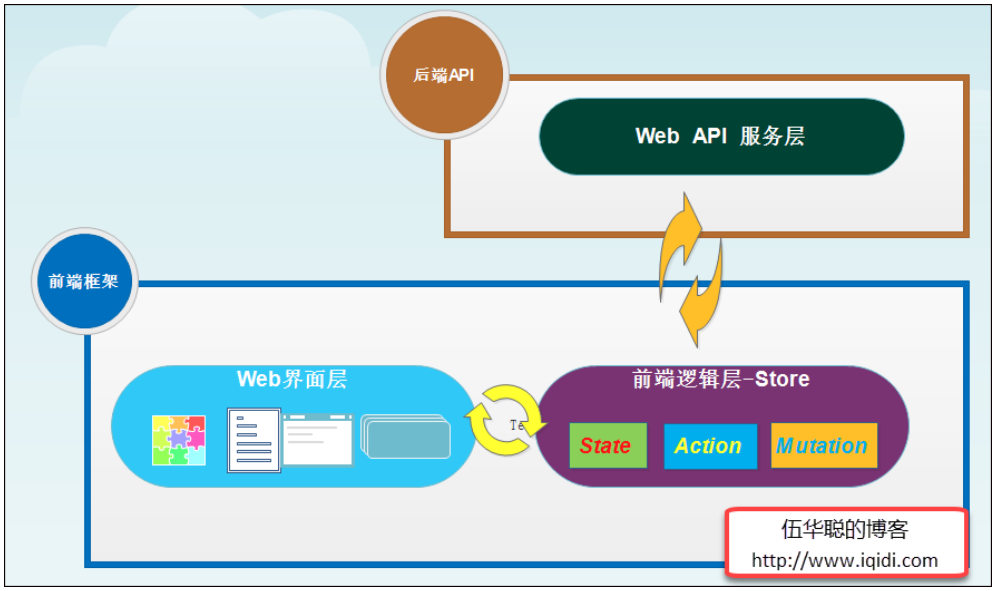
由于倾向于前后端的完全分离，我们后端就可以完全由 Web API 统一构建支持，可以采用.net framework 或者.net core 构建的统一接口平台，可以简单由 Asp.net 做的 Web API 接口平台，也可以基于 ABP-aspnetboilerplate（ABP 框架随笔介绍）框架基础上构建的 Web API 平台。



这样我们就可以基于这些 API 接口构建前端多项应用，如包括 Web 前端、Winform 前端、以及对接各种 APP 等应用。



引入了前后端分离的 VUE + Element 的开发方式，那么前后端的边界则非常清晰，前端可以在通过网络获取对应的 JSON 就可以构建前端的应用了。



在前端处理中，主要就是利用 Vuex 模式中的 Store 对象里实现对 Action 和 Mutation 的请求处理，获取数据后，实现对 State 状态中的数据进行更新。如果仅仅是当前页面的数据处理，甚至可以不需要存储 State 信息，直接获取到返回的数据，直接更新到界面视图上即可。

在开发前期，我们甚至可以不需要和后端发生任何关系，通过 Mock 数据代替从 Web API

上请求数据，只要 Mock 的数据结构和 Web API 接口返回的 JSON 一致，我们就可以在后期实现快速的对接，而不影响现有的代码处理方式。

3.4.6. Axios 网络请求处理

在我们进一步处理前，我们需要知道 Vuex 里面的一些对象概念和他们之间的关系。Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。关于 Vuex 的相关 State、Getter、Mutation、Action、Module 之间的差异和联系，详细可以参考下：https://vuex.vuejs.org/zh/

在开始发起网络请求之前，我们需要了解 axios 这个东西，axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端，本质上也是对原生 XHR 的封装，只不过它是 Promise 的实现版本，符合最新的 ES 规范。在这里我们只需要知道它是非常强大的网络请求处理库，且得到广泛应用即可，列举几个代码案例进行了解。

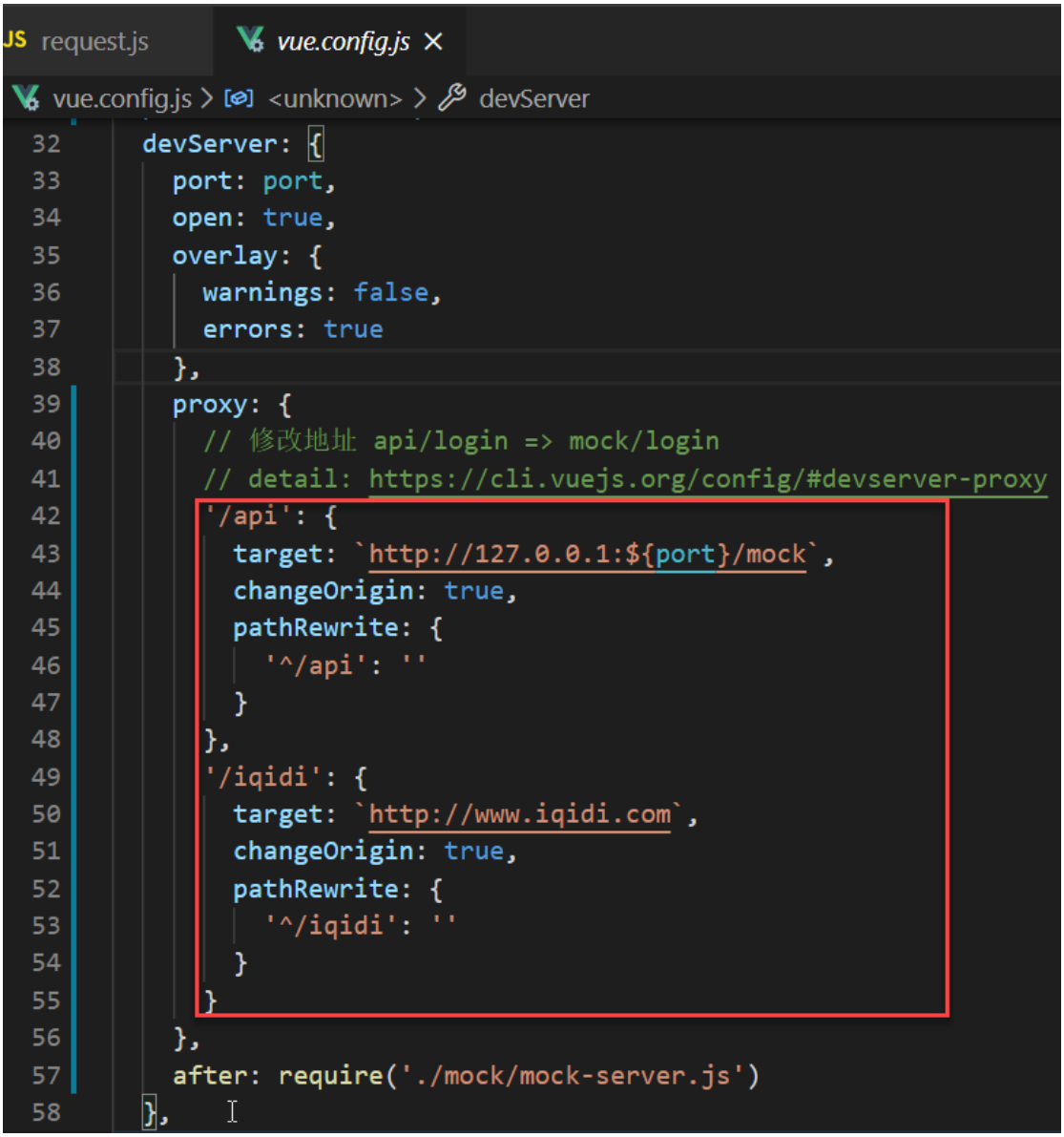
POST 请求：

```
axios({
    method: 'post',
    url: '/user/12345',
    data: {
        firstName: 'Fred',
        lastName: 'Flintstone'
    }
})
.then(function (response) {
    console.log(response);
})
.catch(function (error) {
    console.log(error);
});
```

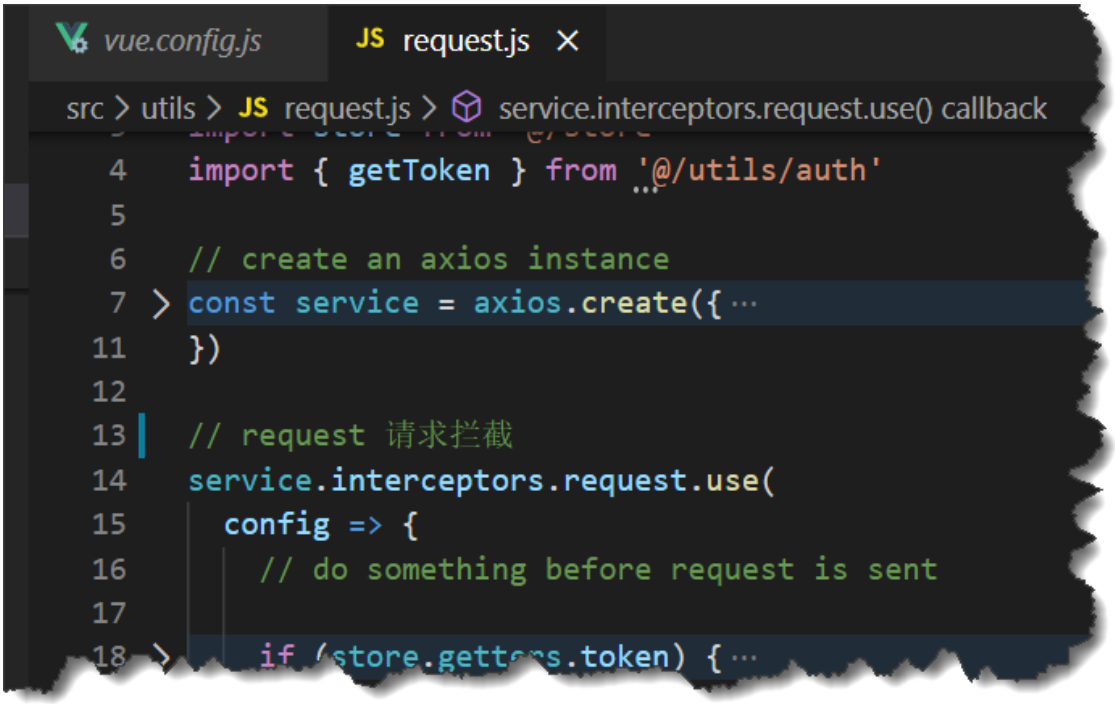
GET 请求：

```
axios
  .get('http://rap2api.taobao.org/app/mock/23080/resources/search',{
     params: {
        id: 5
     }
   })
  .then(res => {
    console.log('数据是:', res);
  })
  .catch((e) => {
    console.log('获取数据失败');
  });
```

如果我们要跨域请求数据，在配置文件里设置代理，vue-cli3 项目，需要在 vue.config.js 里面写配置。



可以分别设置请求拦截和响应拦截，在发出请求和响应到达 then 之前进行判断处理，一般的处理方式就是封装一个类如 request 类，然后进行对拦截器的统一处理，如在请求前增加一些用户身份信息等。


vue.config.js
request.js
src > utils > request.js > service.interceptors.request.use() callback
import { getToken } from '@/utils/auth'
// create an axios instance
const service = axios.create({ ···
})
// request 请求拦截
service.interceptors.request.use(
config => {
// do something before request is sent
if (store.getters.token) {···

```
// create an axios instance
const service = axios.create({
  timeout: 5000 // request timeout
})

// request 请求拦截
service.interceptors.request.use(
  config => {

    if (store.getters.token) {
      config.headers['X-Token'] = getToken()
    }
    return config
  },
  error => {
    // do something with request error
    console.log(error) // for debug
    return Promise.reject(error)
  }
)
```

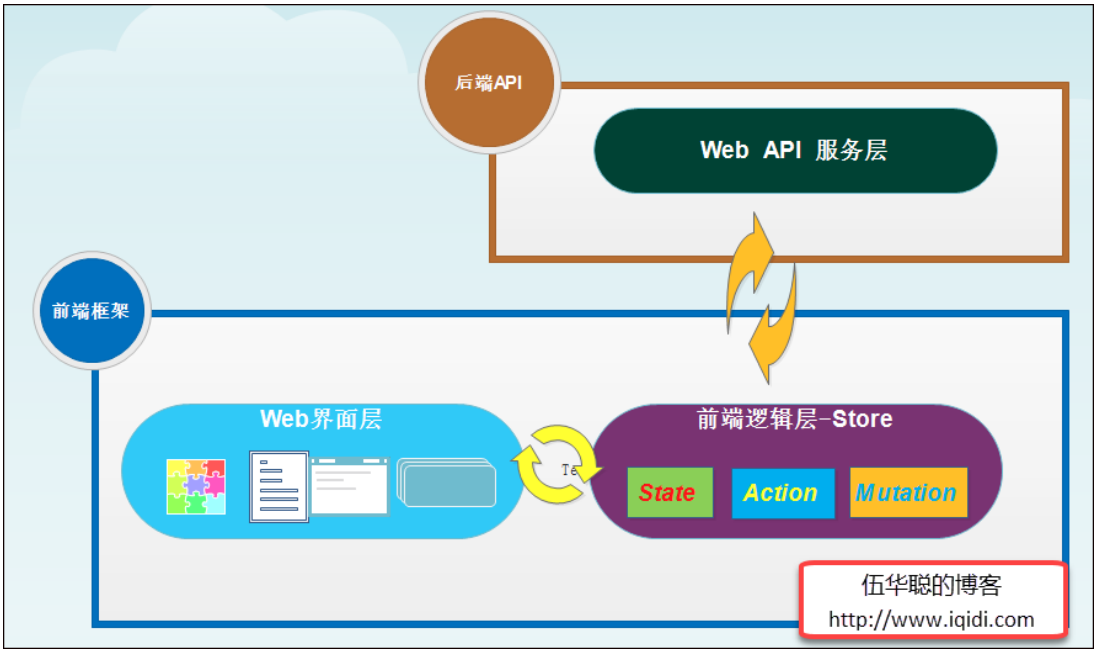
3.4.7. Vuex 中的 API、Store 和 View 的使用

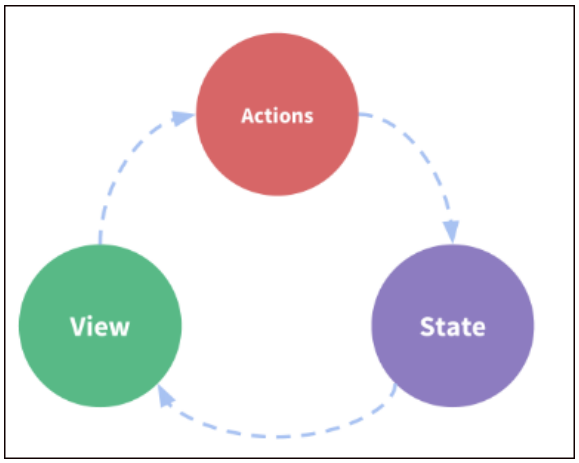
在我们开发 Vue 应用的时候，很多时候需要记录一些变量的内容，这些可以用来做界面状态的承载，也可以作为页面间交换数据的处理，处理这些内容可以归为 Vuex 的状态控制。例如我们往往前端需要访问后端数据，一般是通过封装的 Web API 调用获取数据，而使用 Store 模式来处理相关的数据或者状态的变化，而视图 View 主要就是界面的展示处理。这里主要介绍如何整合这三者之间的关系，实现数据的获取、处理、展示等逻辑操作。

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。关于 Vuex 的

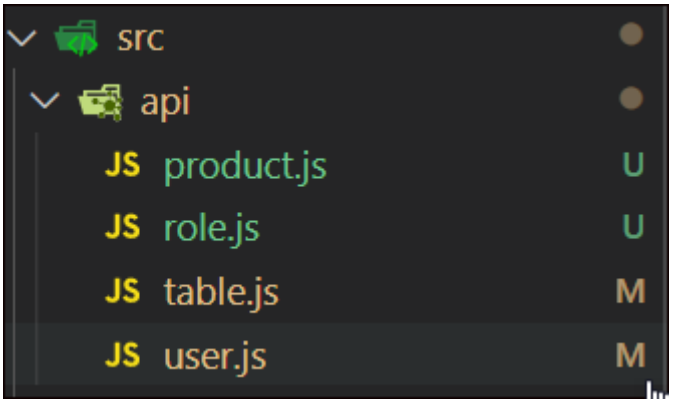
相关 State、Getter、Mutation、Action、Module 之间的差异和联系，详细可以参考下：https://vuex.vuejs.org/zh/ 。

我们再次回到 Vuex 中的 API、Store 和 View 的使用介绍上。





我们来看看 API 的封装请求调用类的封装，如下所示，我们创建了一些操作数据的 API 类文件，每个 API 名称对应一个业务的集中处理，包括特定业务的列表请求、单个请求、增加、删除、修改等等都可以封装在一个 API 类里面。



我们来看看 Product.js 的类文件定义如下所示。

product.js

src > api > product.js > GetProductType

```
// 获取产品类型
export function GetProductType(params) {
  return request({
    url: '/iqidi/h5/GetProductType', //  iqidi => 'http://www.iqidi.com
    method: 'get',
    params
  })
}

// 获取产品列表
export function GetProductList(params) {
  return axios({
    url: '/iqidi/h5/GetProductList', // iqidi => 'http://www.iqidi.com
    method: 'get',
    params
  })
}

// 获取产品明细
export function GetProductDetail(params) {
  return request({
    url: '/iqidi/h5/GetProductDetail', // iqidi =>  'http://www.iqidi.com
    method: 'get',
    params
  })
}
```

这里我用了 Request 和 Axios 的操作对比，两者很接近，因为 request 是对 Axios 的简单封装，主要就是拦截注入一些登录信息和一些响应的错误处理而已。

```
import request from '@/utils/request'
import axios from 'axios'
```

这里的 Url 里面，通过代理配置的处理，会把对应的 iqidi 替换为对应外部域名的处理，从而实现对跨域处理请求数据的获取了，我们这里只需要知道，url 最终会转换为类似 http://www.iqidi.com/h5/GetProductList 这样实际的地址进行请求的即可，返回是一个 JSON 数据集合。

由于 Vue 视图里面的 JS 处理部分，可以直接引入 API 进行请求数据，如下所示。

```
import { GetProductList } from '@/api/product'
```

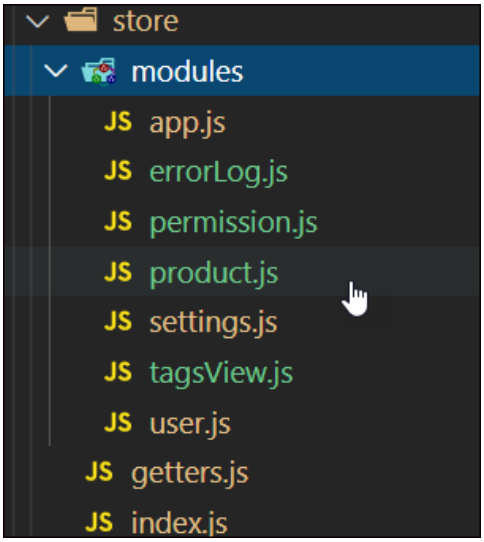
然后我们就可以在 method 方法里面定义一个获取 API 数据的方法了。

```
methods: {
  getlist(type) {
    GetProductList({ type: type }).then(response => {
      const { data } = response
      this.productlist = data.list
      this.listLoading = false
    })
  }
```

这种调用是最直接的 API 调用，没有引入 Store 模块中封装的 Action 或者 Mutation 进行异步或者同步的处理。一般情况下直接使用这种方式比较简洁，因为大多数页面处理或者组件处理，不需要对数据进行全局状态的存储处理，也就是不需要进行全局 Store 对象的处理了。

如果我们需要在全局存储对应的信息，那么就需要引入 Store 模块中对 API 调用的封装了，包括 Action 或者 Mutation 的处理。

我们先来定义 Store 存储类，如下界面所示。



如果我们需要对产品列表等数据进行全局状态的存储，那么我们可以考虑创建一个对应 Store 目录下的模块，如 product.js，来管理 Action、Mutation 和 State 等信息。

src > store > modules > JS product.js > ...

```
import { GetProductList, GetProductDetail } from '@/api/product'

const state = {
  productlist: [],
  productdetail: null
}
const mutations = {
  SET_PRODUCT_LIST: (state, list) => {
    state.productlist = list
  },
  SET_PRODUCT_DETAIL: (state, detail) => {
    state.productdetail = detail
  }
}

const actions = {
  // 产品列表
  getProductList({ commit }, { type }) {
    console.log(type);
    return new Promise((resolve, reject) => {
      GetProductList({ type: type }).then(response => {
        const { data } = response
        commit('SET_PRODUCT_LIST', data)
        resolve(data)
      }).catch(error => {
        reject(error)
      })
    })
  },
```

```
30
31    // 获取产品明细
32    getProductDetail({ commit }, { id }) {
33      return new Promise((resolve, reject) => {
34        GetProductDetail({ id: id }).then(response => {
35          const { data } = response
36          commit('SET_PRODUCT_DETAIL', data)
37          resolve(data)
38        }).catch(error => {
39          reject(error)
40        })
41      })
42    }
43  }
44
45  export default {
46    namespaced: true,
47    state,
48    mutations,
49    actions
50  }
```

我们下来看看，如果引入了 Store 模块的业务类，那么在界面视图中调用代码则修改为调用对应的 Action 或者 Mutation 了。

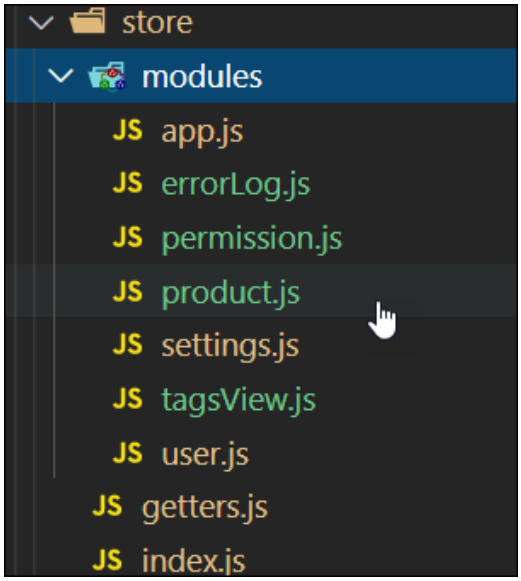
```
methods: {
  getlist(type) {
    // GetProductList({ type: type }).then(response => {
    //   const { data } = response
    //   this.productlist = data.list
    //   this.listLoading = false
    // })

    this.$store.dispatch('product/getProductList', { type: type }).then(data => {
      this.productlist = data.list
      // this.loading = false
    }).catch((e) => {
      // this.loading = false
    })
  }
```

我们这里强调一下，一般情况下在视图模块中使用 API 的类调用即可，不需要累赘的每个业务模块，都创建一个 Store 的模块类进行相应的管理，只有在这些状态数据需要在多

个页面或者组件中需要共享的时候，才考虑引入 Store 模块类进行细化管理。

我们刚才说到，如果需要创建对应业务模块的 Store 状态管理模块，那么需要创建对应的模块类，如前面说到的 product.js 类文件。



其中 Modules 目录里面的按照业务区分边界的 Vuex 的 Store 管理类了，每个对应业务创建一个单独的文件进行管理（如果需要用到的话）。

在 index.js 里面我们通过模块动态加载的方式，把这些类按照不同的命名空间进行加载进来，统一使用。

src > store > index.js > store

```
import Vue from 'vue'
import Vuex from 'vuex'
import getters from './getters'

Vue.use(Vuex)

// https://webpack.js.org/guides/dependency-management/#requirecontext
const modulesFiles = require.context('./modules', true, /\.js$/)

// you do not need `import app from './modules/app'`
// it will auto require all vuex module from modules file
const modules = modulesFiles.keys().reduce((modules, modulePath) => {
  // set './app.js' => 'app'
  const moduleName = modulePath.replace(/^\.\/(.*)\.\w+$/, '$1')
  const value = modulesFiles(modulePath)
  modules[moduleName] = value.default
  return modules
}, {})

const store = new Vuex.Store({
  modules,
  getters
})

export default store

```

4. 其他介绍

4.1. 可重复使用的系统基础模块

.NET 公用类库

俗话说，一个好汉十个帮，众人拾柴火焰高等都说明一个道理，有更多的资源，更丰富的积累，都是助你走向成功，走向顶峰的推动力。就我们开发者而言，其中技巧的积累、资源的积累，就是类似一个个好汉、一根根好柴，是我们能够进行高效开发的保证和推动力。这些类库是我从事多年软件开发，逐渐提炼和发现的一些闪光点或者好片段，有些是吸收别人的优秀的东西，有些是自己逐步提炼的精华，以前，在网络上看到一些开源的项目，总会先看看其是否有封装良好、功能独立的辅助类库，发现好的辅助类库，总是欣喜若狂好一阵子，学习中逐步积累，研究中逐渐提炼，多年过后，略有小成，终为今天所介绍的辅助类库集合。

这些辅助类库平时也并不是所有的都会用得上，不过一些常用的，几乎各个项目就会用

到，类库涉及面非常广，能够为我们开发节省很多时间，并且我们也可以根据自己的需要进行扩充完善，形成自己的类库集合。博客公用类库在线帮助文档列表：

厚积薄发，丰富的公用类库积累，助你高效进行系统开发（1）----开篇总结
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（2）----常用操作
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（3）----数据库相关操作
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（4）----CSV、Excel、INI 文件、独立存储等文件相关
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（5）----热键、多线程、窗体动画冻结等窗体操作
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（6）----全屏截图、图标获取、图片打印、页面预览截屏、图片复杂操作等
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（7）-----声音播放、硬件信息、键盘模拟及钩子、鼠标模拟及钩子等设备相关
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（8）----非对称加密、BASE64 加密、MD5 等常用加密处理
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（9）----各种常用辅助类
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（10）---各种线程同步的集合类
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（11）---各种线程相关操作类
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（12）--- 网络相关操作辅助类
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（13）--- 各种常用的辅助类 2
厚积薄发，丰富的公用类库积累，助你高效进行系统开发（14）---Winform 开发的常用类库

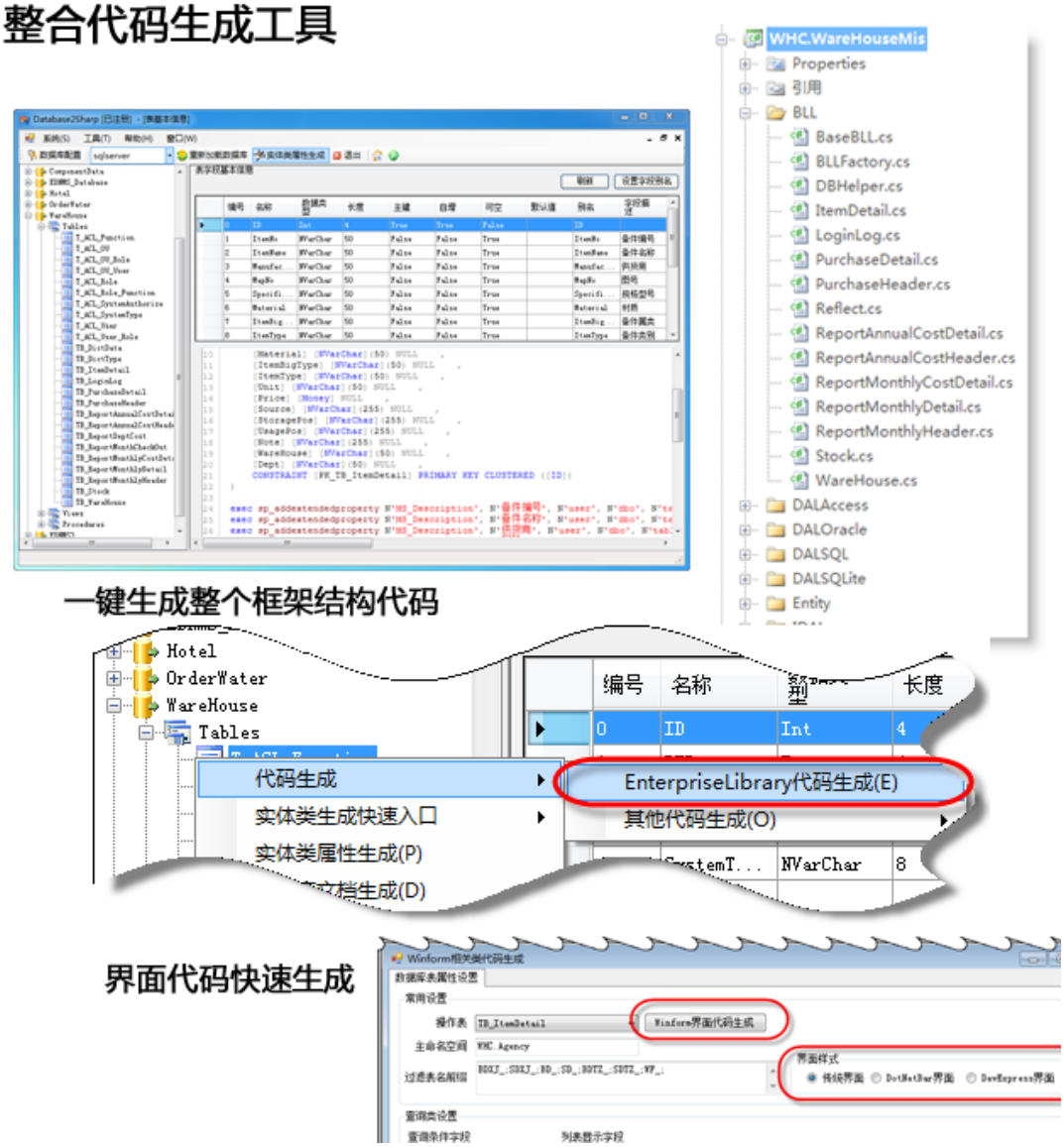
4.2. 代码生成工具 Database2Sharp 的整合

整个框架通过与代码生成工具 Database2Sharp 进行配合，能够一键生成整体性框架代码，Web 开发框架的业务逻辑代码和界面代码，开发更高效。

在整个 Web 开发框架中， Database2Sharp 生成出来的代码体现了非常完美的整合性，能够无缝接入开发的框架系统中，无论是常规的业务逻辑和数据访问层代码，以及列表、编辑界面、添加界面、查看详细界面的 Web 界面代码，都能快速生成，稍作调整即可满足业务模块的需要。

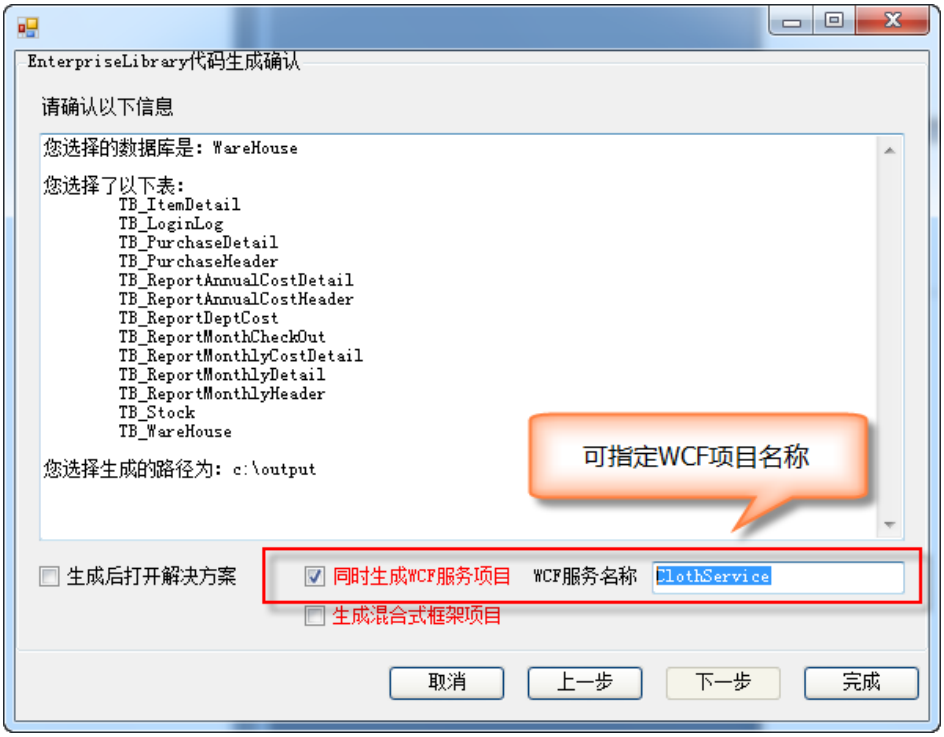
Database2Sharp 是一个简单点击几次鼠标就能完成一周代码量的代码生成工具，效率惊人、友好体贴，真正的开发好伴侣。提供了对 SqlServer 2000/2005/2008、Oracle、Mysql、Access、SQLite 的支持；可以生成各种架构代码以及 Web 界面代码，并且和 Web 开发框架完美整合，体现出更高的开发效率。

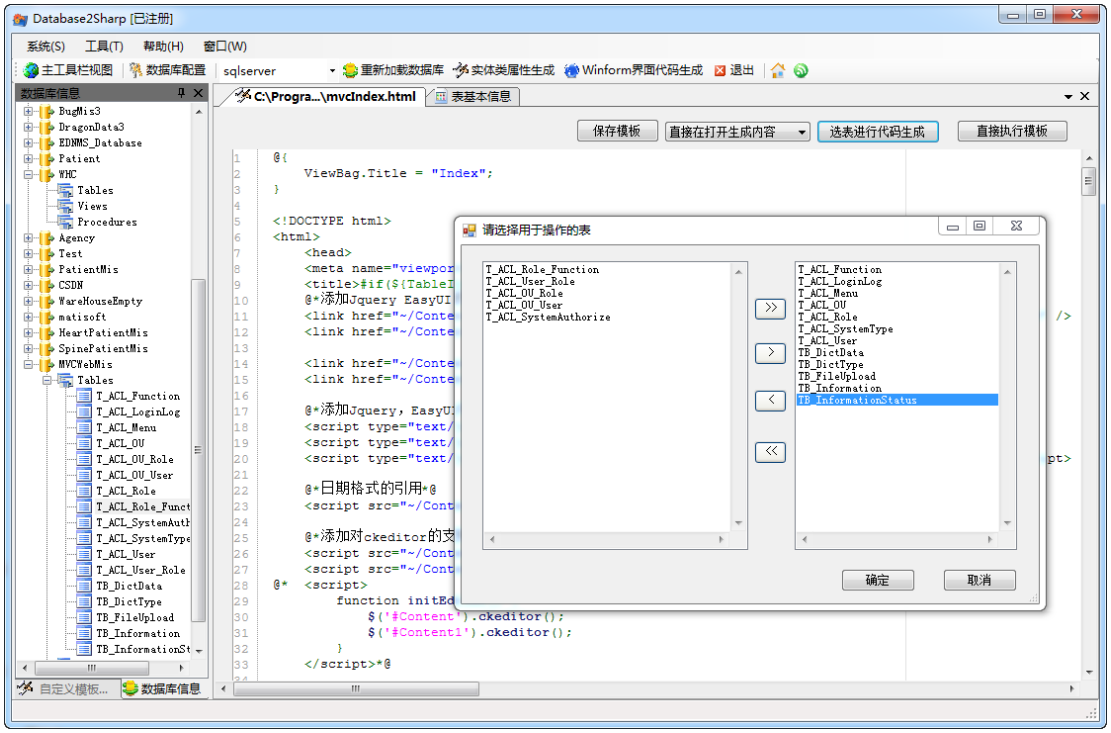
整合代码生成工具



在开发框架中，代码生成工具 Database2Sharp 可以根据需要生成各种不同框架的代码，极大提高了开发效率和企业内部的编码统一。

基于 MVC 的 Web 界面代码，也可以通过我们提供的代码模板文档，在代码工具中运行生成，复制到 Web 界面项目的 View 视图文件夹中即可运行使用。





4.3. 基于多数据库的数据查询模块和通用查询模块

基于多数据库的数据查询模块和通用高级查询模块，查询数据更方便。

在我的 Web 开发框架中，使用了一个查询辅助类 SearchCondition 来实现查询条件的获取和转化，这个辅助类内置了对多种数据库条件的分析处理，因此能够很好生成所需要的数据查询条件，正确高效获取所需的数据进行显示。

```
        /// <summary>
        /// 根据查询条件构造查询语句
        /// </summary>
        private string GetConditionSql()
        {
            //如果存在高级查询对象信息，则使用高级查询条件，否则使用主表条件查询
            SearchCondition condition = advanceCondition;
            if (condition == null)
            {
                condition = new SearchCondition();
                condition.AddCondition("ItemName", this.txtName.Text, SqlOperator.Like)
                    .AddCondition("ItemBigType", this.txtBigType.Text, SqlOperator.Like)
                    .AddCondition("ItemType", this.txtItemType.Text, SqlOperator.Like)
                    .AddCondition("Specification", this.cmbSpecNumber.Text, SqlOperator.Like)
                    .AddCondition("MapNo", this.txtMapNo.Text, SqlOperator.Like)
                    .AddCondition("Material", this.txtMaterial.Text, SqlOperator.Like)
                    .AddCondition("Source", this.txtSource.Text, SqlOperator.Like)
                    .AddCondition("Note", this.txtNote.Text, SqlOperator.Like)
                    .AddCondition("Manufacture", this.txtManufacture.Text, SqlOperator.Like)
                    .AddCondition("ItemNo", this.txtItemNo.Text, SqlOperator.LikeStartAt)
                    .AddCondition("WareHouse", this.txtWareHouse.Text, SqlOperator.Like)
                    .AddCondition("Dept", this.txtDept.Text, SqlOperator.Like)
                    .AddCondition("UsagePos", this.txtUsagePos.Text, SqlOperator.Like)
                    .AddCondition("StoragePos", this.txtStoragePos.Text, SqlOperator.Like);
            }
            string where = condition.BuildConditionSql().Replace("Where", "");
            return where;
        }
```

```
    /// <summary>
    /// Sql 的查询符号
    /// </summary>
    public enum SqlOperator
    {
        [Description("Like 模糊查询")]
        Like,
        [Description("Not LiKE 模糊查询")]
```

```
        NotLike,
        [Description("Like 开始匹配模糊查询，如 Like 'ABC%'")]
        LikeStartAt,
        [Description("= 等于号")]
        Equal,
        [Description("<> (≠) 不等于号")]
        NotEqual,
        [Description("> 大于号")]
        MoreThan,
        [Description("<小于号")]
        LessThan,
        [Description("≥大于或等于号 ")]
        MoreThanOrEqual,
        [Description("≤ 小于或等于号")]
        LessThanOrEqual,
        [Description("在某个字符串值中")]
        In
    }
```

另外，结合上面的多数据库的数据查询模块，基于 MVC 的 Web 界面，我们可以设计一个通用的分页条件查询操作。

先在客户端的绑定相关条件的值，如果是数值、日期类型的，那么是一个区间的值，用波浪符号~进行分开，并且每个参数以一个特殊的前缀开始，如 WHC_，如下代码所示。

```
//绑定搜索按钮的的点击事件
function BindSearchEvent() {
    $("#btnSearch").click(function () {
        //字段增加WHC_前缀字符，避免传递如URL这样的Request关键字冲突
        var queryData = {
            WHC_SystemType_ID: $("#txtSystemType_ID").combobox('getValue'),
            WHC_LoginName: $("#txtLoginName").val(),
            WHC_FullName: $("#txtFullName").val(),
            WHC_Note: $("#txtNote").val(),
            WHC_IPAddress: $("#txtIPAddress").val(),
            WHC_MacAddress: $("#txtMacAddress").val(),
            WHC_LastUpdated: $("#txtLastUpdated").datebox('getValue') + "~" + $("#txtLastUpdated2").datebox('getValue')
        }
        //将值传递给
        InitGrid(queryData);
        return false;
    });
}
```

在控制器处理端，我们把分页查询的操作，封装在了业务基类 Business<B，T>这个类上面，可以为继承于它的业务控制器类提供了通用的分页查询。下面是控制器基类获取传递过来的参数并构造查询条件的代码。

```
/// <summary>
/// 获取分页操作的查询条件
/// </summary>
protected virtual string GetPagerCondition()
{
    #region 根据数据库字段列，对所有可能的参数进行获值，然后构建查询条件
    SearchCondition condition = new SearchCondition();
    DataTable dt = baseBLL.GetFieldTypeList();
    foreach (DataRow dr in dt.Rows)
    {
        string columnName = dr["ColumnName"].ToString();
        string dataType = dr["DataType"].ToString();
        //字段增加WHC_前缀字符，避免传递如URL这样的Request关键字冲突
        string columnValue = Request["WHC_" + columnName] ?? "";

        if (IsDateTime(dataType))
        {
            condition.AddDateCondition(columnName, columnValue);
        }
        else if (IsNumericType(dataType))
        {
            condition.AddNumberCondition(columnName, columnValue);
        }
        else
        {
            condition.AddCondition(columnName, columnValue, SqlOperator.Like);
        }
    }
    #endregion

    return condition.BuildConditionSql().Replace("Where", "");
}
```

由于我一直希望我的 Web 开发框架能够精益求精，所以设计了这个通用查询模块，希望使用该 Web 开发框架的客户，能够快速、高效地实现数据的查询。

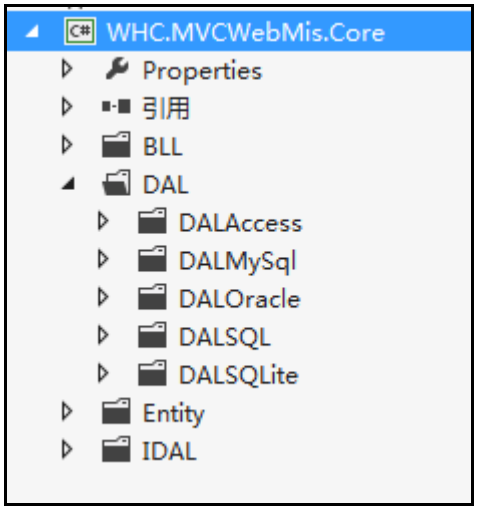
4.4. 框架提供基于多种数据库的整合

框架提供基于多种数据库（Sqlserver/Oracle/Mysql/Sqlite/Access)的整合。

虽然我们在实际项目中，一般采用一种数据库进行处理，但是不同的项目，采用的数据库类型可能不同，本 Web 开发框架为了方便演示和扩展的需要，内置支持了 Sqlserver/Oracle/Mysql/Sqlite/Access，更多的数据库，也可以通过扩展数据库访问基类的方式进行更多数据库的支持。

Web 开发框架里面的所有模块，如用到了数据存储的，如权限管理管理模块、通用数据字典管理模块，均内置支持这几种数据库的整合支持。整个 Web 开发框架的数据库访问，

能够手动配置数据库类型，对于同一种数据库，也可以把数据存储分开存储，如业务数据存储在一个数据库，权限管理控制存储在另外一个数据库这种方式。



Web 开发框架提供多种数据库支持，但数据访问基类依然很精简，因为我们利用的数据库访问模块是 Enterprise Library，把数据库抽象化，并且我把所有数据库通用操作放在了一个超级基类上，具体的数据库基类只需要实现变化的部分即可，业务访问类则使用泛型进行封装处理。

因此，Web 开发框架提供了高度封装的数据访问基类，开发代码更少更高效。

4.5. 特性总结

Web 开发框架，本身就是为了能够快速开发一个高效、稳定、美观大方、扩展性强的应用软件系统。因此我在自己十年左右的共享软件开发生涯以及公司项目开发中，不断思考，精雕细琢，对很多重要的特性都进行了归纳和升华，吸收项目中好的闪光点，借鉴一些好的软件开发思路，力求把软件做的更好。

在开发效率方面，除了开发一些常规通用的模块、在模块内部又充分考虑继承、重用的规则，还对大幅度提高效率的代码生成工具，根据 Web 开发框架的实现思路和特点，进行了完善优化，使得无论在业务代码生成，还是在界面代码生成方面，均能把开发效率发挥到极致，希望整个 Web 开发框架能够持续发挥它的魅力和吸引力，为更多的人带来希望，体验开发的乐趣。