实验报告

任务

基于第一个项目爬虫爬取的数据（3-5个数据源），完成数据展示网站。
基本要求：
1、用户可注册登录网站，非注册用户不可登录查看数据
2、用户注册、登录、查询等操作记入数据库中的日志
3、实现查询词支持布尔表达式 （比如“新冠 AND 肺炎”或者“新冠 OR 肺炎”）
4、爬虫数据查询结果列表支持分页和排序
5、用Echarts或者D3实现3个以上的数据分析图表展示在网站中
扩展要求（非必须）：
1、实现对爬虫数据中文分词的查询
2、实现查询结果按照主题词打分的排序
3、用Elastic Search+Kibana展示爬虫的数据结果

由于最近时间太紧了，权衡下，只选择完成最基本的要求了。

在期中的大作业中，我已经完成了腾讯，网易和搜狐这三个网页的信息爬取和结构化存储。在这里就直接使用数据库中的这些数据，不作新的数据采集工作。

之前作业的链接：

https://blog.csdn.net/nbyvy/article/details/116300934?spm=1001.2014.3001.5501

功能一、登入登出与注册

首先是用户的注册登入登出操作，那么就需要在后台的后端的数据库中维护一张用户信

息表 user。

建表操作

```
--创建用户信息数据表
CREATE TABLE `crawl`.`user` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL,
`password` VARCHAR(45) NOT NULL,
`registertime` datetime DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
UNIQUE KEY `username_UNIQUE` (`username`))
ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

用了 username，password，registertime 这三个属性。

后端实现

位于 public/routes/users.js 中。

注册实现函数：

```
/* add users */
router.post('/register', function (req, res) {
var add_user = req.body;
// 先检查用户是否存在
userDAO.getByUsername(add_user.username, function (user) {
if (user.length != 0) {
// res.render('index', {msg:'用户不存在！'});
res.json({msg: '用户已存在！'});
}else {
userDAO.add(add_user, function (success) {
res.json({msg: '成功注册！请登录'});
})
}
});


});
```

通过路由/register 接受前端请求数据，前端会传来的如下 json 格式的数据：

```
var data = JSON.stringify({
    username: $scope.add_username,
    password: $scope.add_password
});
```

接收到数据后，通过 userDAO.getByUsername,连接数据库的 user 表，并且查询 username 与传来数据相匹配的数据。

这是我们实现在文件夹 dao 中写好的函数。userDAO.js 负责连接数据库中的 user 表，并且定义了两个函数：

```
add: function (user, callback) {
pool.query(userSqlMap.add, [user.username, user.password],
function (error, result) {
```

```
if (error) throw error;
callback(result.affectedRows > 0);
});
},
getByUsername: function (username, callback) {
pool.query(userSqlMap.getByUsername, [username], function
(error, result) {
if (error) throw error;
callback(result);
});
},
```

一个是为了方便向数据库中添加元组的 add，和一个通过 username 查询元组的 getByUsername。这样以后就很方便的调用该模块中的两个函数来执行 sql 语句。

回到 users.js

在查询数据库之后会做一次判断，如果数据库返回结果为空，那就返回注册成功，执行 add 操作，向数据库中写入注册的用户信息；

如果数据库返回不为空，那么就输出一条（“用户已存在”），不作其他处理。

登录实现函数

```
router.post('/login', function(req, res) {
var username = req.body.username;
var password = req.body.password;
// var sess = req.session;
```

```
userDAO.getByUsername(username, function (user) {
if(user.length==0){
res.json({msg:'用户不存在！请检查后输入'});
}else {
if(password===user[0].password){
req.session['username'] = username;
res.cookie('username', username);
res.json({msg: 'ok'});
// res.json({msg:'ok'});
}else{
res.json({msg:'用户名或密码错误！请检查后输入'});
}
}
});
});
```

通过/login 路由接受前端传来的如下 json 格式数据：

```
var data = JSON.stringify({
    username: $scope.username,
    password: $scope.password
});
```

、

接受到数据之后调用 getByUsername 函数，查询数据库的相应记录。如果没有查到，那么就表明用户数据库中没有该用户信息，不能登录成功。

查到记录之后还需要再便定用户输入的密码和查询到的密码是不是一致，如果一致，那么登录成功并且将用户的 username 存入 cookie，计入 session 数组中；

如果不一致那么就是没有登录成功。

登出实现函数

```
// 退出登录
router.get('/logout', function(req, res, next){
```

```
// 备注：这里用的 session-file-store 在 destroy 方法里，并没有销毁 cookie
// 所以客户端的 cookie 还是存在，导致的问题 --> 退出登陆后，服务端检测到 cookie
// 然后去查找对应的 session 文件，报错
// session-file-store 本身的 bug

req.session.destroy(function(err) {
if(err){
res.json('退出登录失败');
return;
}
// req.session.loginUser = null;
res.clearCookie('username');
res.json({result:'/index.html'});
});
});
```

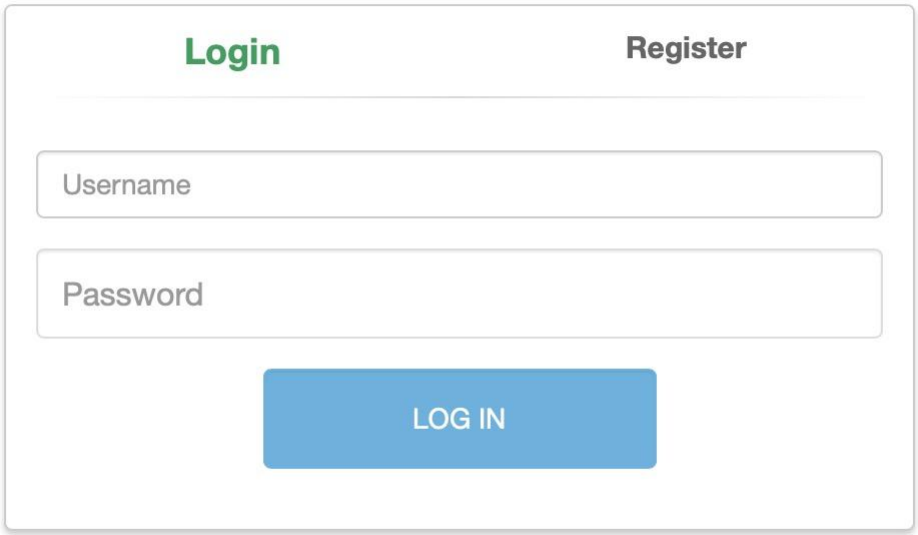
路由/logout 接收到登出请求后，只需要清空 session 和 cookie 就行了。

前端的实现

用的是老师给的参考模版。

除了一些 html 元素和 css 样式之外，还用到了一些脚本，比如输入不能为空，组册页面两次密码的输入必须一致，满足条件后向后端特定的路由传递数据等。

这里代码方面不作详细解读。


Login
Register
Username
Password
LOG IN

功能二、用户操作日志

就是将用户的请求的信息写入数据库中。

因此还需要建立一张表：

```
--记录用户的登陆，查询（具体查询语句）操作
CREATE TABLE `crawl`.`user_action` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL,
`request_time` VARCHAR(45) NOT NULL,
`request_method` VARCHAR(20) NOT NULL,
`request_url` VARCHAR(300) NOT NULL,
`status` int(4),
`remote_addr` VARCHAR(100) NOT NULL,
PRIMARY KEY (`id`))
ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

记录用户名，请求时间，请求的方式，请求的链接，请求的状态，远程用户等信息。

同样在 dao 中的 logDAO 中，连接了数据库中的 user_action 表，并且定义了向数据库添加元组的 userlog 函数，以便后期调用快速实现 sql 语句的输入：

```
userlog :function (useraction, callback) {
pool.query('insert into
user_action(username,request_time,request_method,request_
url,status,remote_addr) values(?, ?,?,?,?,?)',
useraction, function (error, result) {
if (error) throw error;
callback(result.affectedRows > 0);
});
},
```

实现该功能的后端代码位于 app.js 中：

```
let method = '';
app.use(logger(function (tokens, req, res) {
console.log('打印的日志信息：');
var request_time = new Date();
var request_method = tokens.method(req, res);
var request_url = tokens.url(req, res);
var status = tokens.status(req, res);
var remote_addr = tokens['remote-addr'](req, res);
if(req.session){
var username = req.session['username']||'notlogin';
}else {
var username = 'notlogin';
}

// 直接将用户操作记入 mysql 中
if(username!='notlogin'){
logDAO.userlog([username,request_time,request_method,requ
est_url,status,remote_addr], function (success) {
console.log('成功保存！');
})
}
```

每次的请求的会传入：

通过 new Date()存储每次的接受到请求的时间；

通过 tokens，存储请求的状态、方法、链接和远程用户；

通过 session，存储 username；

最终通过调用之前就定义好的 uselog 的方法，直接使用 sql 的 insert 语句存入数据库。

此外，日志存数据库后，后台还将日志信息打印出来（代码比较简单，不作展示了）：

```
打印的日志信息:
请求时间  =  2021-06-29T06:41:22.049Z
请求方式  =  GET
请求链接  =  /news/pie
请求状态  =  200
请求长度  =  undefined
响应时间  =  1.986ms
远程地址  =  ::1
远程用户  =  undefined
http版本  =  1.1
浏览器信息 =  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) Ap
ersion/14.1 Safari/605.1.15
用户 =  zgl111
 ================
成功保存!
```

功能三、查询

主要是实现关键词以及逻辑连接词的组合查询。

先上前端页面：

向输入框中输入要查询的关键词，并且在 AND 和 OR 中选择一个，进行查询。

这个通过输入的内容来选择相应的 sql 语句来来实现。在 dao 中的 newsDAO.js 中定义 search 函数来构建相应的 sql 语句：

```
search :function(searchparam, callback) {
// 组合查询条件
var sql = 'select * from fetches ';

if(searchparam["t2"]!="undefined"){
sql +=(`where title like '%${searchparam["t1"]}%'
${searchparam['ts']} title like '%${searchparam["t2"]}%'
`);
}else if(searchparam["t1"]!="undefined"){
sql +=(`where title like '%${searchparam["t1"]}%' `);
};
if(searchparam["t1"]=="undefined"&&searchparam["t2"]=="un
defined"&&searchparam["c1"]!="undefined"){
sql+='where ';
```

```
}else
if(searchparam["t1"]!="undefined"&&searchparam["c1"]!="un
defined"){
sql+='and ';
}
if(searchparam["c2"]!="undefined"){
sql +=(`content like '%${searchparam["c1"]}%'
${searchparam['cs']} content like '%${searchparam["c2"]}%'
`);
}else if(searchparam["c1"]!="undefined"){
sql +=(`content like '%${searchparam["c1"]}%' `);
}
if(searchparam['stime']!="undefined"){
if(searchparam['stime']=="1"){
sql+='ORDER BY publish_date ASC ';
}else {
sql+='ORDER BY publish_date DESC ';
}
}
sql+=';';
```

大致的原理就是：

不断判定搜索框中的字段是不是为空；

如果不是的就从数据库中对应的属性中查询。

快速构建 sql 之后，就是调用了。具体的实现在 routes/new.js 中：

```
outer.get('/search', function(request, response) {
console.log(request.session['username']);
//sql 字符串和参数
if (request.session['username']===undefined) {
// response.redirect('/index.html')
response.json({message:'url',result:'/index.html'});
}else {
var param = request.query;
newsDAO.search(param,function (err, result, fields) {
response.json({message:'data',result:result});
})
}
});
```

就是从/search 路由中接受前端传来的数据，并且调用 search 函数查询并且返回。

功能四、查询结果分页展示

这个是在前端文件 search.html 文件实现的。当然分页展示，在网上都是有很多模版的。

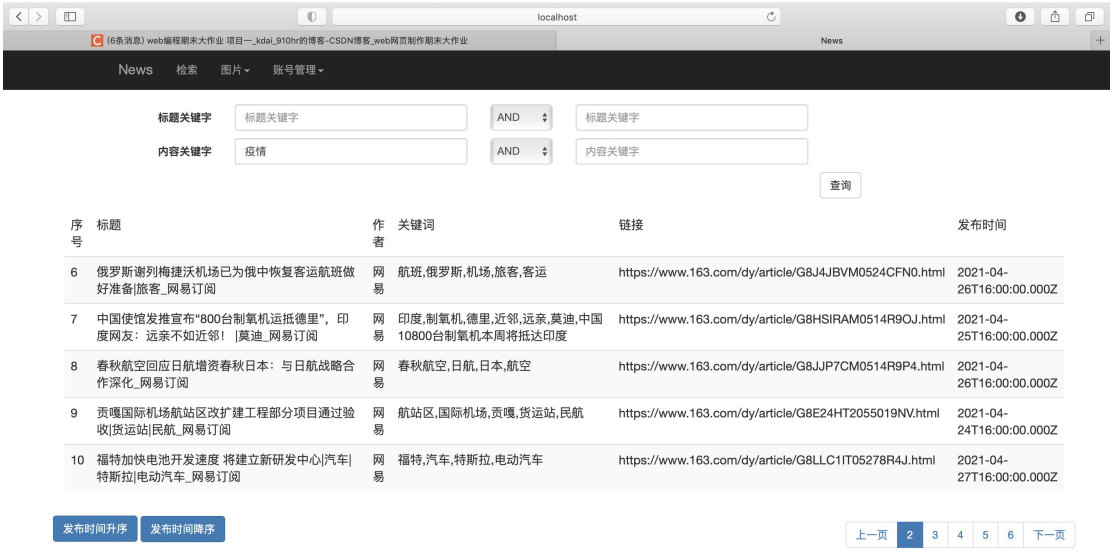
后端查询结果得到后返回前端。

```
<table class="table table-striped">
<thead>
<tr>
<td>序号</td>
<td>标题</td>
<td>作者</td>
<!-- <td>内容</td>-->
<td>关键词</td>
<td>链接</td>
<td>发布时间</td>
</tr>

</thead>
<tbody>
<tr ng-repeat="(key, item) in items">
<td>{{index+key}}</td>
<td>{{item.title}}</td>
<td>{{item.author}}</td>
<!-- <td>{{item.content}}</td>-->
<td>{{item.keywords}}</td>
<td>{{item.url}}</td>
<td>{{item.publish_date}}</td>
</tr>
</tbody>
</table>
```

将返回的元素全部放置于 table 中。

```
<a ng-click="selectPage(page)" >{{ page }}</a>
```

通过这一条语句分页展示出来。其中的脚本函数在 public/javascript/new.js 中。

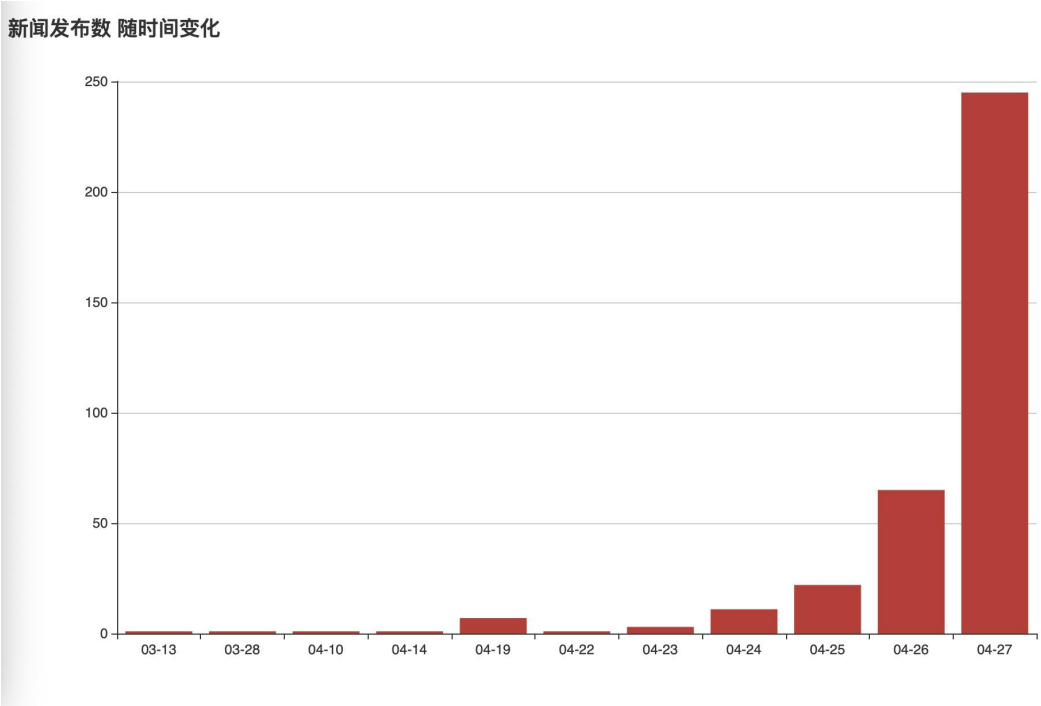


按时间排序就在数据库查询是判定一下是不是需用按照时间生序降序就好了：

```
if(searchparam['stime']!="undefined"){
if(searchparam['stime']=="1"){
sql+='ORDER BY publish_date ASC ';
}else {
sql+='ORDER BY publish_date DESC ';
}
```

功能五、可视化

柱状图（新闻数量-日期）



前端 news.html 点击后，启动脚本。Histogram()将请求发送给后端：

```
<li><a ng-click="histogram()">柱状图</a></li>
<li><a ng-click="pie()">饼状图</a></li>
<li><a ng-click="line()">折线图</a></li>
<li><a ng-click="wordcloud()">词云</a></li>
```

后台的代码位于 routes/new.js 中：

```
router.get('/histogram', function(request, response) {
//sql 字符串和参数
console.log(request.session['username']);

//sql 字符串和参数
if (request.session['username']===undefined) {
// response.redirect('/index.html')
response.json({message:'url',result:'/index.html'});
}else {
var fetchSql = "select publish_date as x,count(publish_date)
as y from fetches group by publish_date order by
publish_date;";
newsDAO.query_noparam(fetchSql, function (err, result,
fields) {
response.writeHead(200, {
"Content-Type": "application/json",
"Cache-Control": "no-cache, no-store, must-revalidate",
"Pragma": "no-cache",
"Expires": 0
});
response.write(JSON.stringify({message:'data',result:resu
lt}));
response.end();
});
}
});
```

在路由/histogram 中接受前端的请求。

接受到请求之后，从数据查询 publish_date 与其计数，并且返回给前端:

```
select publish_date as x,count(publish_date) as y from fetches group by publish_date order by publish_date;
```

前端接受到数据之后，在 histogram()中绘制柱状图。

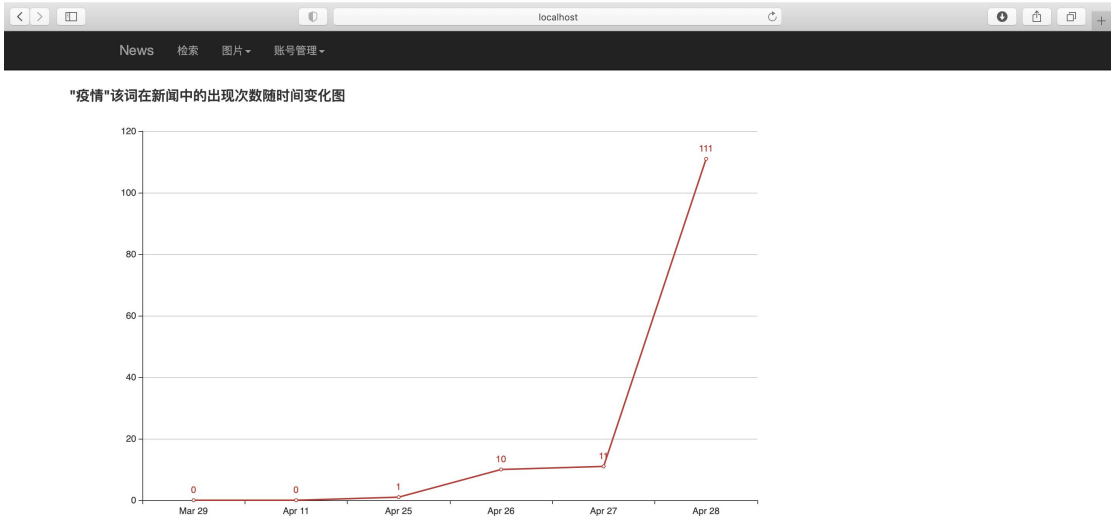
调用了 echarts 可以方便绘制多种图像。

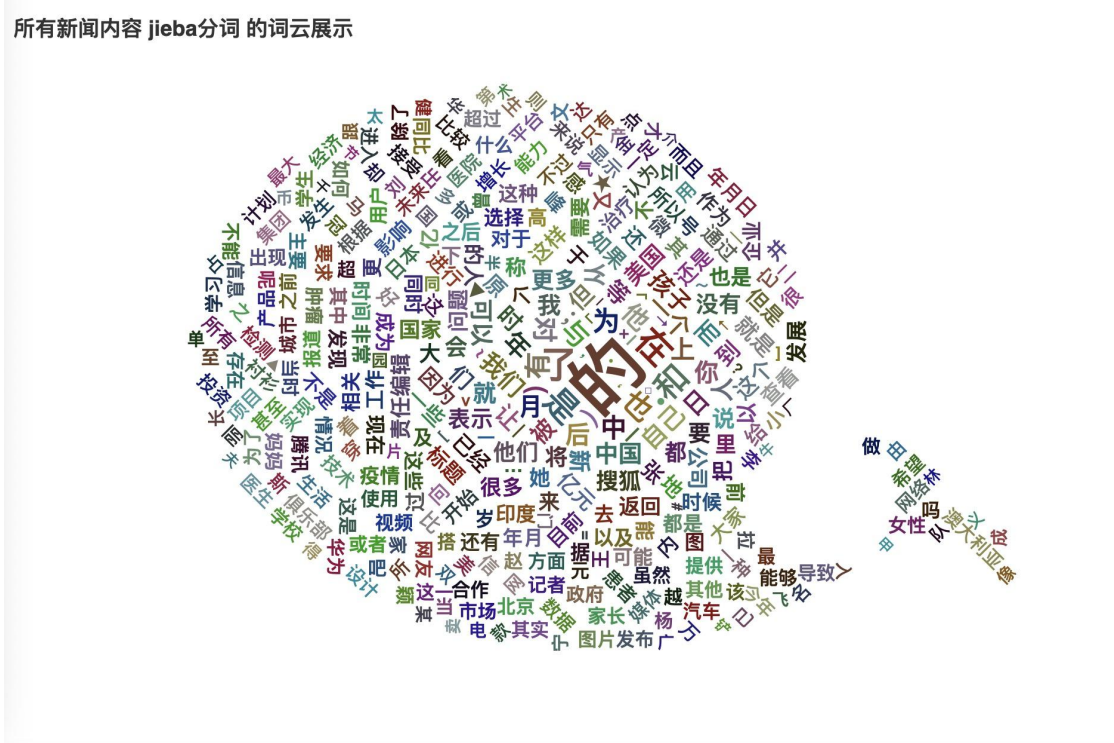
```
$scope.histogram = function () {
$scope.isShow = false;
$http.get("/news/histogram")
.then(
function (res) {

if(res.data.message=='url'){
window.location.href=res.data.result;
}else {
// var newdata = washdata(data);
let xdata = [], ydata = [], newdata;
var pattern = /\d{4}-(\d{2}-\d{2})/;
res.data.result.forEach(function (element) {
// "x":"2020-04-28T16:00:00.000Z" ,对 x 进行处理,只取 月日
xdata.push(pattern.exec(element["x"])[1]);
ydata.push(element["y"]);
});
newdata = {"xdata": xdata, "ydata": ydata};
var myChart =
echarts.init(document.getElementById('main1'));
// 指定图表的配置项和数据
var option = {
title: {
text: '新闻发布数 随时间变化'
},
```

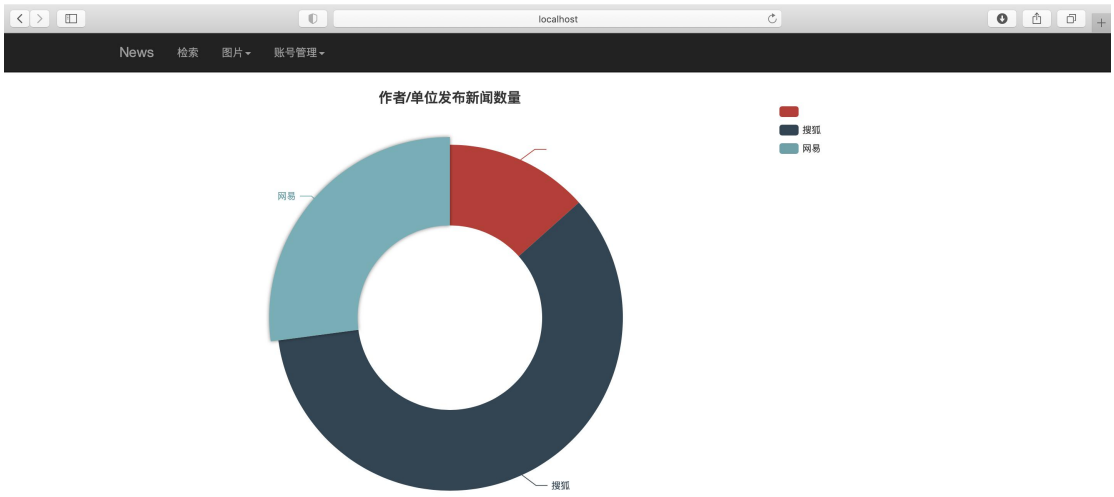
```
tooltip: {},
legend: {
data: ['新闻发布数']
},
xAxis: {
data: newdata["xdata"]
},
yAxis: {},
series: [{
name: '新闻数目',
type: 'bar',
data: newdata["ydata"]
}]
};
// 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);
}
},
function (err) {
$scope.msg = err.data;
});
};
```

同理,折线图，饼状图，词云的机制大致类似,其中用到了 jieba 分词，词频统计等函数实现，都是套用了参考模版，不作详细的展示了。

直接上图展示：







总结

至此所有的基本功能都实现了。由于本人的拖延症作祟，导致到大作业和期末考试全挤在了一起，很遗憾没能有时间精力继续完成扩展功能。首先反省，下次一定调整好进度！！

前后端分离的软件架构。在 web 编程的学习过程，我最有体会的就是了解到前后端分离的软件架构，包括了前端怎么和后端 nodejs 传输数据，前端的使用 css 样式和 js 脚本，前端接受后端的数据并且展示，后端处理请求接受并且返回数据等等。只能说刚刚入门。

自我感觉，js 语法是我目前学习的语言中最难的了。虽然基本的语法，比如基本数据类型循环、选择结构，函数调用等是编程中比较通用的也是比较好上手，但是 js 也有自己具有特色的难点，比如函数的异步执行和嵌套使用，数据的路由，前端的 js 脚本等等。

这次实现之后也算是用来之后写一些简易的前后端的网页 app。