法律文书检索系统实验报告

周启恒-16337334 郑育聪-16337329

简介

法律案例的文书对法律行业工作者有较大的参考价值，目前市场上的法律文书检索工具较少，主要为**中国裁判文书网、无讼、威科、理脉**等检索工具，其中所有的文书数据都来源于**中国裁判文书网**。

在对现有的检索工具进行初步调研后，我们发现其检索功能存在一些缺点，如——

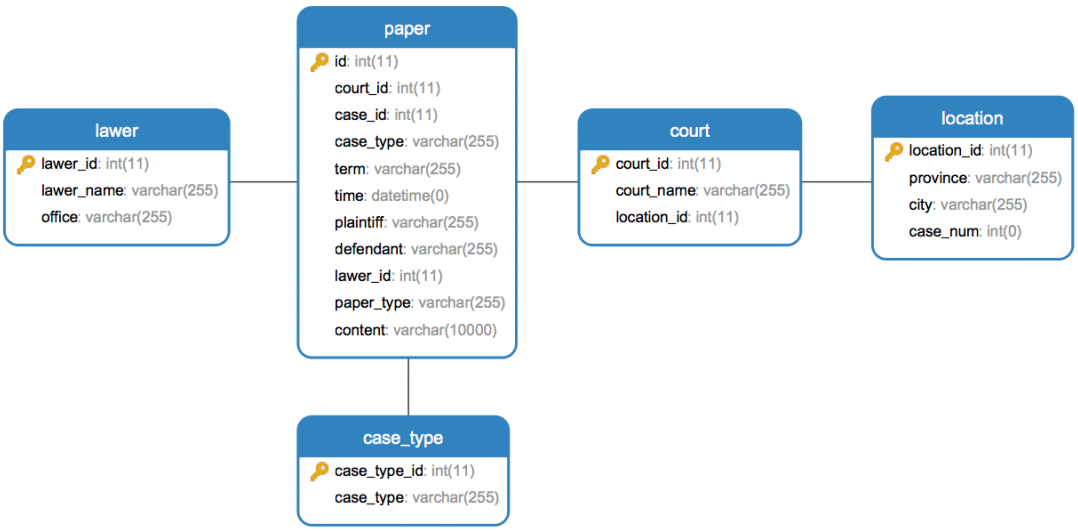
- 只能支持精确匹配，即搜索内容必须是文书中“精确”出现的内容才会被检索到，如搜索“送外卖 狗 咬”可以检索到结果，但搜索“送外卖被狗咬”无法检索到结果
- 检索结果不够精确，如搜索“下雨 不可抗力”，排在前面的很多只出现了“不可抗力”一个关键词
- 不支持句式搜索，即无法进行关键词之间的“且、或、非”搜索

为此，本次实验我们小组实现一个较为简单的**法律文书检索系统**，并在查询功能上做出针对上述问题的改进。

数据库构建

关系表的构建

根据我们本次实验的需要，我们构建了5个实体，分别为：裁判文书、案件类型、法院、地区、律师，对应的E-R图如下：



按照E-R图转化成逻辑结构的规则，每个实体转换成一个关系，多对多的联系也转换成一个关系。因此，根据上述 E-R 图设计数据库各关系逻辑结构如下：

- **文书paper**

```
#paper表 --- id,法院id，标题，案号，案件类型（民事、刑事等），审理阶段（一审、二审等），
#             时间，原告，被告，文书类型（裁定、判决等），文书内容
CREATE TABLE `paper` (
    `id` int(11) NOT NULL,
    `court_id` int(11) NULL,
    `title` varchar(255) NULL,
    `case_id` varchar(255) NULL,  -- 案号，字符串类型
    `case_type` varchar(255) NULL,
    `term` varchar(255) NULL,
    `time` datetime NULL,
    `plaintiff` varchar(255) NULL,
    `defendant` varchar(255) NULL,
    `paper_type` varchar(255) NULL,
    `content` varchar(20000) NULL,
     PRIMARY KEY (`id`) );
```

- **案件类型case_type**

```
#case_type表 --- 案件类型id，案件类型（民事、刑事等）
CREATE TABLE `case_type` (
    `case_type_id` int(11) NOT NULL,
    `case_type` varchar(255) NULL,
    PRIMARY KEY (`case_type_id`));
```

- **法院court**

```
#court表 --- 法院id，法院名称，所在地区id
CREATE TABLE `court` (
    `court_id` int(11) NOT NULL,
    `court_name` varchar(255) NULL,
    `location_id` int(11) NULL,
    PRIMARY KEY (`court_id`));
```

- **地区location**

```
#location表 --- 地区id，省，市，案件数量
CREATE TABLE `location` (
    `location_id` int(11) NOT NULL,
    `province` varchar(255) NULL,
    `city` varchar(255) NULL,
    `case_num` int NULL,
    PRIMARY KEY (`location_id`));
```

- **律师lawer**

```
#lawer表 --- 律师id，律师姓名，律师所在事务所
CREATE TABLE `lawer` (
    `lawer_id` int(11) NOT NULL,
    `lawer_name` varchar(255) NULL,
    `office` varchar(255) NULL,
    PRIMARY KEY (`lawer_id`));
```

数据获取

本次实验中的数据通过网络爬虫爬取中国裁判文书网的数据获取，中国裁判文书网上文书数量大，为了方便，本次实验中只爬取了近6000份文书，并对爬取到的数据按照上述表及属性提取对应的数据。

简单标签的提取

- 裁判文书网上的数据偏结构化，部分简单的标签可使用**正则表达式直接匹配**即可获得，如文书标题、发布日期、省、市等，**以日期为例**，提取过程如下：
  - 观察网站返回的HTML数据，可以发现日期紧跟在`PubDate`后面，使用RE直接匹配即可
  - 使用`pandas`的`apply`函数对所有文书进行处理，并去除极少数没有日期标签的数据

```
#文书时间
def get_pu_date(x):
    res = re.findall(r'PubDate"":""(.*?)"', x)
    res = '' if len(res) == 0 else res[0]
    return res
wenshu_data['pu_date'] = wenshu_data['raw_data'].apply(lambda x:
get_pu_date(x))
wenshu_data =
wenshu_data[wenshu_data['pu_date']!=''].reset_index(drop=True)
```

复杂标签的提取

- 有些复杂标签，如**原告、被告、双方代理人**均没有单独的标签，且由于中文的复杂性，同一意思可以有多种不同的表达方式，如**原告可以表达为上诉人、申请人、起诉人**等。以原告、被告、代理人的提取为例：
  - 由于文书的规范性，其所在位置是固定的，即**原告、被告、双方代理人**会出现在文书的开头，如下设置足够的规则进行提取，可以提取出大部分所需数据

```
def get_yuangao_agent_beigao_agent_one(each): #提取原告、原告律师、被告、被告律
师
    res = ['', '', '', ''] #原告、原告律师、被告、被告律师
    i = 0
    while i < len(each): #原告部分
        if re.match(r'上诉人|原告|申请.*?人|起诉人|...', each[i]) == None:
break
        res[0] += re.findall(r'(上诉人|原告|申请.*?人|起诉人|...)(\(.*?\)|
(.*?) |, ){0,1}(.*?)[。 , ]', each[i])[0][2] + ' '
        i += 1
    while i < len(each): #原告代理人部分
```

```
            if '代理人' not in each[i]:
                if re.match(r'被', each[i]) == None:
                    i += 1
                    continue
                else: break
            res[1] += re.findall(r'代理人(.*)', each[i])[0] + ' '
            i += 1
        /*被告与被告代理人同样设置相应的规则进行提取，此处略去*/
```

以上为法律文书的数据提取，其余关系表可根据法律文书提取到的数据进行构建，此处省略。

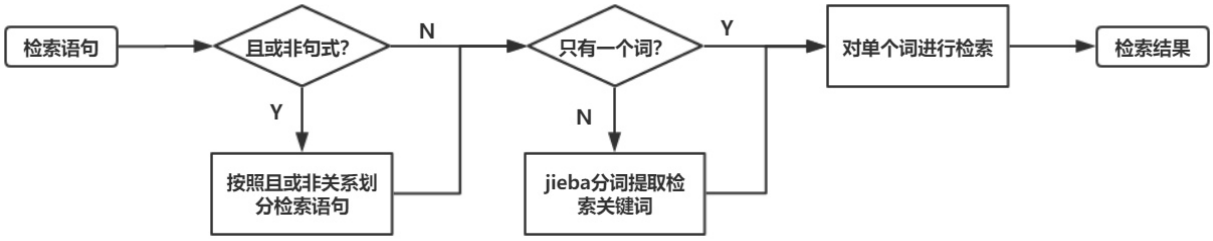
数据导入

本次数据导入分为3个部分——

- 使用pymysql连接mysql，根据上述关系表编写相应mysql语句，调用`cursor.execute()`**构建关系表**
- 编写mysql相应的约束语句，增加关系之间的外键约束，调用`cursor.execute()`执行
- 使用`load data infile`指令导入数据，并调用`conn.commit()`提交，将导入结果写入磁盘中

查询实现

针对已有工具的缺点，我们实现如下几种情况的查询——**对单个词的检索、句式搜索、分词检索**，并构造整个检索过程如下：



具体的实现如下：

对单个词的检索

- 对单个词的搜索，如地点、法院名称、案件类型、标题、年份、文书内容，分为以下步骤：
  - 到地点`location`中进行搜索，若该词出现表中，则根据得到的`location_id`在文书`paper`表中进行查询得到结果，若未出现，则进行接下来的检索
  - 到法院`court`中进行搜索，规则同上（若存在，则可获取结果作为输出，否则继续搜索），对案件类型`case_type`和文书标题`title`也做同样的操作
  - 若该词未出现在上述关系表中，则直接根据**文书内容**进行查找，以其查找结果作为最终的输出

```
def search_exact_match_one_word(query): #query是一个词，地点、法院名称、案件类型、标题、年份、文书内容，只能精确匹配
    place_res = search_location(query)
    if place_res != None: return place_res
    court_res = search_court(query)
    if court_res != None: return court_res
    case_type_res = search_case_type(query)
    if case_type_res != None: return case_type_res
    title_res = search_title(query)
    if title_res != None: return title_res
    return search_in_content(query)
```

- 其中，以按照地点`location`进行搜索为例，包括以下几个部分——
  - 编写mysql查询语句在`location`中使用`like`语句匹配词是否在省份或者城市属性中
  - 若存在，则编写查询语句查询到对应地区的所有文书

```
def search_location(place): #若place在location中，返回文书结果，否则返回空集
    sql = """SELECT location_id
             FROM location
             WHERE province like '%%%s%%' or city like '%%%s%%' """ %
(place,place)
    res = cursor.execute(sql)
    if res: #地点可以匹配到，则根据地点搜索
        location_id = cursor.fetchall() #元组的元组
        location_id = list(zip(*location_id))[0] #转为单个元组
        sql = """SELECT * FROM paper
                 WHERE court_id in (SELECT court_id FROM court
                                    WHERE location_id in %s""" %
str(location_id)[:-2]+'));' #去掉元组最后的逗号，加上结尾
        cursor.execute(sql)
        return cursor.fetchall()
    return None
```

句式搜索

- 本次实验中实现简单的“且、或、非”句式搜索，具体如下——
  - **非**：先根据`^`后面的词在文书内容中检索到结果，再使用全部的文书调用`minus`函数去除掉这些结果
  - **或**：根据`|`进行划分，再对划分的结果分别查询，调用`add`函数求并集作为结果
  - **且**：根据`&`进行划分，再对划分的结果分别查询，调用`intersect`函数求交集作为结果

```
def search(query):#与或非句式搜索
    if query[0] == '^': #非
        query = query[1:].strip()
        return minus(search_title(''),
search_in_content(query[1:].strip()))
```

```
    if '|' in query: #或
        queries = query.split('|')
        a = []
        for each in queries:
            a = add(a, search_exact_match_one_word(each.strip()))
        return a
    if '&' in query: #且
        queries = query.split('&')
        a = search_title('')
        for each in queries:
            a = intersect(a, search_exact_match_one_word(each.strip()))
        return a
```

- 其中，以`minus`函数为例，求两个结果的差集，只需使用简单的列表生成式即可

```
def minus(all, A): #结果的差集
    res = [each for each in all if each not in A]
    return res
```

分词检索

针对其他工具只能精确匹配的缺点，我们的模糊匹配实现如下：

- 使用`jieba`中文分词库对检索语句进行分词，若只有一个词，则根据上述**对单个词的检索**进行检索
- 否则，使用`jieba`的`posseg`包进行分词并获取词性，只对`n*:名词,v:动词，z：状态`进行检索
- 对检索到的结果，求交集排在前面，做并集排在后面，使得前面的文书是匹配到较多关键词的

```
def cut_word_match(query): #进行分词模糊匹配
    query = query.strip()
    word_flag_ls = list(psg.cut(query))
    if len(word_flag_ls) == 1: return search_exact_match_one_word(query)
    tmp_res = []
    res = search_title('')
    for pair in word_flag_ls:
        pair = list(pair)
        if pair[1].startswith('n') or pair[1].startswith('v') or
pair[1].startswith('z'): #名词、动词、状态
            tmp = search_exact_match_one_word(pair[0]) #每个词的搜索结果
            tmp_res.append(tmp)
            res = intersect(res, tmp) #取交集，排在前面
    for tmp in tmp_res:
        [res.append(each) for each in tmp if each not in res]
    return res
```

网站构建

网站结构：tree图

```
├── law_paper_system
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
├── manage.py
└── system
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── forms.py
    ├── migrations
    |   ├── 0001_initial.py
    |   ├── 0002_casetype_court_lawer_location.py
    |   ├── __init__.py
    |   └── __pycache__
    |       ├── 0001_initial.cpython-36.pyc
    |       ├── 0002_casetype_court_lawer_location.cpython-36.pyc
    |       └── __init__.cpython-36.pyc
    ├── models.py
    ├── search_func.py
    ├── static
    |   └── css
    |       └── layouts
    |           ├── blog-old-ie.css
    |           └── blog.css
    ├── templates
    |   ├── base.html
    |   ├── login.html
    |   ├── main_window.html
    |   ├── paper_all.html
    |   ├── paper_detail.html
    |   └── paper_edit.html
    ├── tests.py
    ├── urls.py
    └── views.py
```

构建步骤：

- 创建django项目，连接本地数据库mysql，添加自己的app（system）
- 同步mysql数据库：将数据库中的table转换为django的models
- 添加url，views，html文件

  原理：

  - url是一个域名，对应一个页面。url于views函数相关联，当用户访问一个url时，即给views函数发送一个request（可能带有参数）。

    - 在对应的views中进行判断、操作，这里可能包含对数据库进行增删查改多种操作。在本项目中主要调用队友用py写好的search接口，处理好数据，将其传给html文件。
    - html接受views传入的参数，在前端按照一定的样式对传入的数据进行渲染、显示，从而给用户美观、清晰的用户界面，便于阅读与一系列后续的操作。
- 添加forms.py，其中存储可能用到的表单类，规范其格式。

关键代码的展示：

- models：关系在django中的存储形式：

```
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#   * Rearrange models' order
#   * Make sure each model has one field with primary_key=True
#   * Make sure each ForeignKey has `on_delete` set to the desired
behavior.
#   * Remove `managed = False` lines if you wish to allow Django to create,
modify, and delete the table
# Feel free to rename the models, but don't rename db_table values or field
names.
from django.db import models


class Admin(models.Model):
    username = models.CharField(primary_key=True, max_length=25)
    password = models.CharField(max_length=25, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'Admin'


class CaseType(models.Model):
    case_type_id = models.IntegerField(primary_key=True)
    case_type = models.CharField(max_length=255, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'case_type'


class Court(models.Model):
    court_id = models.IntegerField(primary_key=True)
    court_name = models.CharField(max_length=255, blank=True, null=True)
    location = models.ForeignKey('Location', models.DO_NOTHING, blank=True,
null=True)

    class Meta:
        managed = False
```

```
        db_table = 'court'


class Lawer(models.Model):
    lawer_id = models.IntegerField(primary_key=True)
    lawer_name = models.CharField(max_length=255, blank=True, null=True)
    office = models.CharField(max_length=255, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'lawer'


class Location(models.Model):
    location_id = models.IntegerField(primary_key=True)
    province = models.CharField(max_length=255, blank=True, null=True)
    city = models.CharField(max_length=255, blank=True, null=True)
    case_num = models.IntegerField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'location'


class Paper(models.Model):
    id = models.IntegerField(primary_key=True)
    court = models.ForeignKey(Court, models.DO_NOTHING, blank=True,
null=True)
    title = models.CharField(max_length=255, blank=True, null=True)
    case_id = models.CharField(max_length=255, blank=True, null=True)
    case_type = models.CharField(max_length=255, blank=True, null=True)
    term = models.CharField(max_length=255, blank=True, null=True)
    time = models.DateTimeField(blank=True, null=True)
    plaintiff = models.CharField(max_length=255, blank=True, null=True)
    defendant = models.CharField(max_length=255, blank=True, null=True)
    paper_type = models.CharField(max_length=255, blank=True, null=True)
    content = models.CharField(max_length=20000, blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'paper'
```

- url：对应不同页面的域名。

```
from django.conf.urls import url
from . import views

#url包含一个域名，一个view，一个name
urlpatterns = [
    url(r'^$', views.main_window, name='main_window'),
    url(r'^paper/(?P<paper_id>[0-9]+)/$', views.paper_detail,
name='paper_detail'),
    url(r'^paper_add/$', views.paper_add, name='paper_add'),
    url(r'search/$', views.search, name='search'),
    url(r'login/$', views.login, name='login'),
    url(r'delete/(?P<paper_id>[0-9]+)/$', views.delete, name='delete'),
]
```

- views.main_window：用于显示主界面，利用search_one_api搜索第一个文书，用于示例的展示，返回main_window.html。

```
def main_window(request):
    paper = search_one_api(1)
    paper[10] = paper[10][:8]
    return render(request, 'main_window.html', {'paper': paper})
```

- main_window.html：扩展base.html
  - 增加搜索栏
  - 接收一个文书对象，并显示。
  - 显示网站关于：数据来源，制作者

```
{% extends 'base.html' %}

{% load staticfiles %}
{% block content %}

<div class="content pure-u-1 pure-u-md-3-4">
    <form class="pure-form" method="GET" action="search">
            <fieldset>
                <h1 class="post-title">搜索文书</h1>
                <input class="pure-input-3-4" type="search" name='q'
placeholder="输入关键词">
                <button type="submit" class="pure-button pure-button--
group">搜索</button>
                <button class="pure-button pure-button--hover"
style="float: right;"><a href="{% url 'paper_add' %}">上传文书</a></button>
            </fieldset>
    </form>
    <br><br>
    <div class="brand-title">示例:</div>
```

```
    <div class="posts">
            <h1 class="content-subhead">文书序号:{{paper.0}}</h1>
            <section class="post">
                <div class="pure-u-md-11-12">
                <header class="post-header">
                    <div><h2>标题: {{paper.2}}</h2></div>
                    <p class="post-meta">
                        <a class="post-category post-category-design">时间:
</a> <a class="post-category post-category-pure">文书性质:</a>
                    </p>
                </header>
                </div>

                <div class="post-description">
                    {% for i in paper.10 %}
                    <p>{{i}}</p>
                    {% endfor %}
                </div>
                <button class="pure-button" style="float: right;"><a href="
{% url 'paper_detail' paper_id=paper.0 %}">文书详情</a></button>
            </section>
    </div>
    <br><br>
    <div class="posts">
            <h1 class="content-subhead">关于</h1>
            <section class="post">
                    <div class="post-description">
                        <p>
                            <h3>A simple legal instrument database system
for database course.</h3>
                            <h3>Source of data:<a
href="https://wenshu.court.gov.cn/"> 中国裁判文书网.</a></h3>
                        </p>
                    </div>
            </section>
    </div>
    <div class="footer">
        <a>By ZYC ZQH</a>
    </div>

</div>

{% endblock %}
```

- views.paper_add：用于提交新的文书，具体执行步骤分为两部分
    - 初次点击时，是get请求，需要创建一个表单对象，传给html文件，显示一个表单页面，用于填写。

    - 第二次submit时，是post请求，接收前一个页面中表单的数据，创建新的model，并保存。之后redirect到一个paper_detail的页面，用于显示刚提交的表单的详情。

```
def paper_add(request):
    #如果是保存操作
    if request.method == 'POST':
        form = paperForm(request.POST)
        if form.is_valid():
            paper = form.save(commit=False)
            paper.save()
            return redirect('paper_detail', paper.id)
    else:
        form = paperForm()
    return render(request, 'paper_edit.html', {'form':form})
```

- paper_add.html

```
{% extends 'base.html' %}
{% block content %}

<div class="content pure-u-1 pure-u-md-3-4">
<form method="POST" class="pure-form pure-form-aligned">
    {% csrf_token %}
        <h1>提交新的文书</h1>
        <div class="pure-control-group">
        {{form.as_p}}
        </div>
        <button style="float:right" type="submit" class="pure-button pure-
button-primary">Submit</button>
    </form>
</div>
{% endblock %}
```

- views.paper_detail：根据paper_id，利用自己写好的api在数据库中进行搜索，并做一些格式化的处理，之后将paper对象传给html文件。

```
#可作为一个文书的详细信息显示页面
def paper_detail(request, paper_id):
    try:
        paper = search_one_api(paper_id)
        #return render(request, 'paper_detail.html', {'paper': paper})
    except Exception as identifier:
        tmp_paper = get_object_or_404(Paper, id=paper_id)
        paper = search_one_api(1).copy()
        paper[0] = tmp_paper.id
        paper[2] = tmp_paper.title
```

```
            paper[6] = tmp_paper.time
            paper[9] = tmp_paper.paper_type
            paper[10] = [tmp_paper.content]
        return render(request, 'paper_detail.html', {'paper': paper})
```

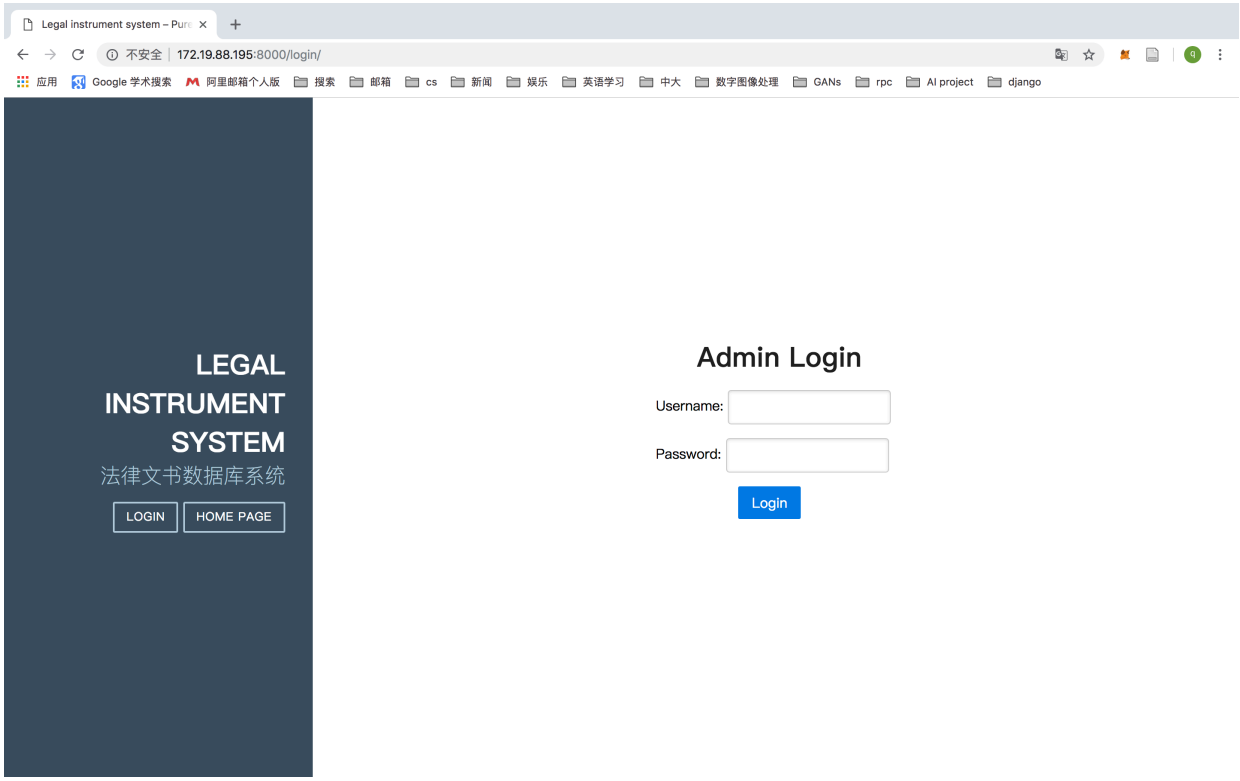
- 接收views传来的paper对象，按格式显示其具体属性，利用for循环格式化显示文书详情。

```
{% extends 'base.html' %}
{% block content %}
{% load staticfiles %}

<div class="content pure-u-1 pure-u-md-3-4">
    <div class="posts">
        <h1 class="content-subhead">文书序号:{{paper.0}}</h1>
        <section class="post">
            <div class="pure-u-4-5">
            <header class="post-header">
                    <div><h2>标题: {{paper.2}}</h2></div>
                <p class="post-meta">
                    <a class="post-category post-category-design">时间:
{{paper.6}}</a> <a class="post-category post-category-pure">文书性质:
{{paper.9}}</a>
                </p>
            </header>
            </div>

            <div class="post-description">
                {% for i in paper.10 %}
                <p>{{i}}</p>
                {% endfor %}
            </div>
        </section>
    </div>
</div>
{% endblock %}
```

实验结果

- 系统网页主界面：

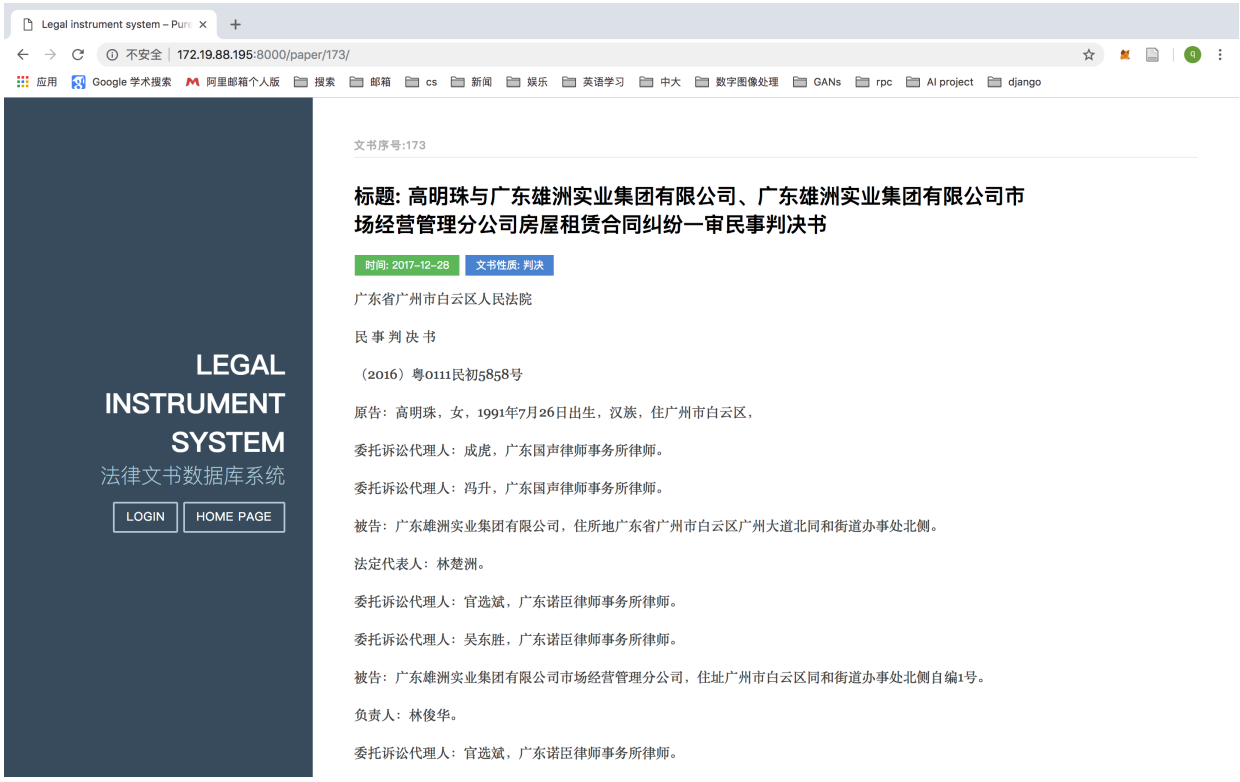
- 登录功能：



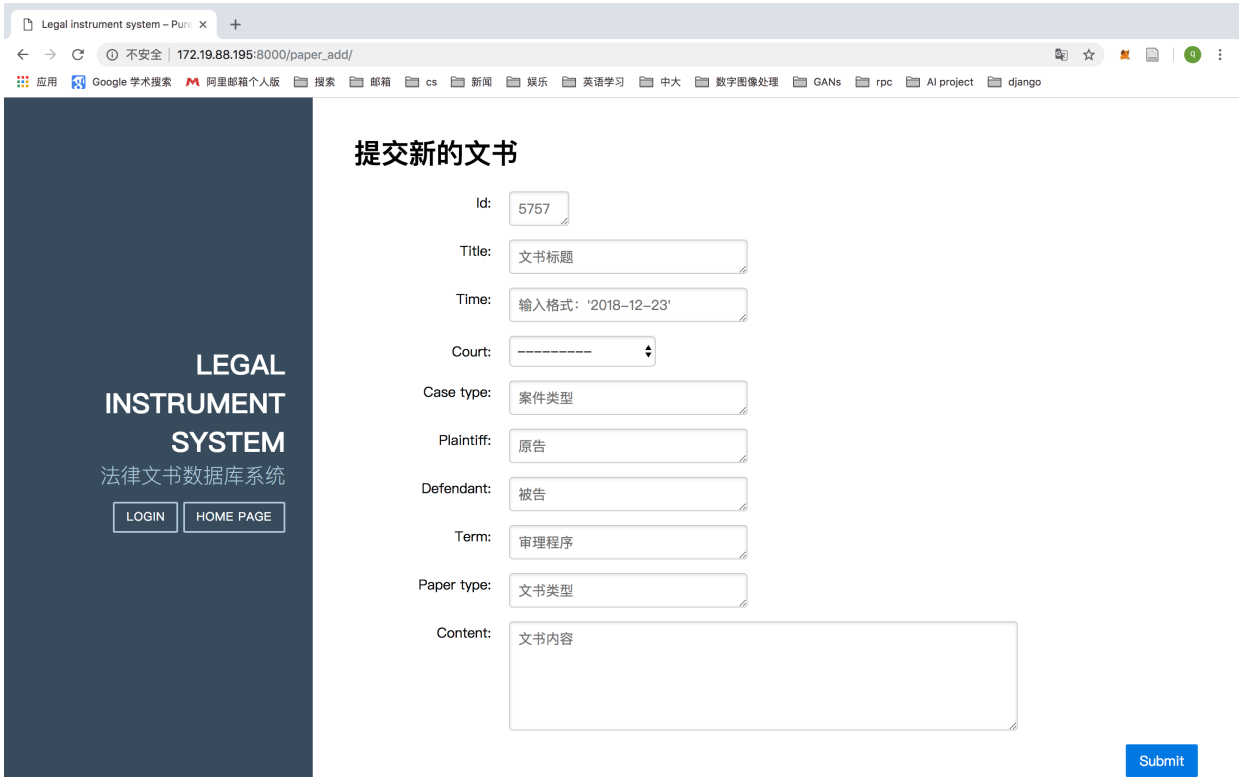
- 搜索功能（示例：广州）：
  - 显示包含关键字的所有记录（概述）
  - 可点击删除文书、文书详情

- 并操作搜索（示例：江苏&离婚）

- 文书详情页面（文书id173）



- 添加文书：



实验工具与分工

本次实验中，使用工具如下：

- 使用Mysql作为数据库
- Python使用爬虫相关库爬取数据
- Python使用pymysql与数据库进行连接和操作
- 使用Django作为web后端框架，前端HTML、CSS对数据显示进行渲染。

实验分工如下：

周启恒：负责E-R图及对应关系表的构造，负责整个网站的搭建，并调用队友提供的接口实现查询等功能。

郑育聪：负责根据关系表的属性获取、提取数据，将数据导入数据库，并编写查询等接口供队友调用。