司法搜索引擎系统的设计与实现

熊天翼　　严澜
计 95　　建 82
2019011303　　2018010005

1 需求描述

根据给定的中文法律数据集，实现一个司法搜索引擎系统。

• 核心功能：用户界面，关键词检索，案例详情监视。

• 其他功能：高级检索，关键词高亮，标签显示与搜索，基于法条搜索，类似案例推荐，关键词高级检索，以案搜案。

2 框架设计

- **前端**：Vue 框架 + Element UI，基于色色引擎 UI 实现。
  https://github.com/YunYouJun/sese-engine-ui
- **后端** Django + Whoosh + sqlite3

3 核心模块设计与功能实现

3.1 数据的导入与存储

用 xml.etree.ElementTree 库对 xml 文件进行解析，提取全文、文首、当事人、法律条文、案件基本情况、裁判分析过程、判决结果、文尾、文书名称、审判程序、法庭、审判理由、省份、年份共十四个字段，使用 tree.getroot().find('.//xxxx').get('value') 找到相应字段。剔除不满足上述要求的法律案件，总共成功从 68417 个法律案例中导入了 50548 个案件。

在 django 中，针对案件（Case）和法条（Law）分别建立模型，对应于 sqlite3 数据库中的两张数据表。模型对应的字段如下

```
class Law(models.Model):
    id = models.AutoField(primary_key=True) # 法律id
    name = models.CharField(max_length=50) # 法条名称（在导入数据时保证不出现重复）
    def __str__(self) -> str:
        return self.name

class Case(models.Model):
    id = models.AutoField(primary_key=True)
    qw_value = models.TextField(max_length=500) # 全文
```

```
    head = models.CharField(max_length=200) # 文首
    related_people = models.TextField() # 当事人
    judicial_record = models.TextField() # 诉讼记录
    basic_info = models.TextField() # 案件基本情况
    judgement_process = models.TextField() # 判决分析过程
    result = models.TextField() # 判决结果
    tail = models.TextField() # 文尾
    note_name = models.CharField(max_length=30) # 文书名称
    judge_prop = models.CharField(max_length=30) # 审判程序（一审/二审）
    court = models.CharField(max_length=30) # 经办法院
    case_reason = models.CharField(max_length=30) # 案由
    province = models.CharField(max_length=200) # 行政区划-省份
    year = models.IntegerField(default=2023) # 年份
    laws = models.ManyToManyField(to=Law, related_name="cases") # 法律条文
```

由于一个案件中可能与多个法条相关，且同一法条可能出现在多个案件中，因此我们使用多对多的映射关系（ManyToManyField）进行存储和查询。

3.2 使用 Whoosh 框架建立索引与分词

使用 Whoosh 作为搜索引擎框架，Whoosh 是一个用 Python 编写的全文搜索引擎库，支持倒排索引的自动建立，提供简单易用的 API，支持快速搜索、排序和定制。

使用 Whoosh 进行中文分词时，需导入 Whoosh 源代码 whoosh_backend_ZW.py，替换部分不兼容包，并将 analyzer 替换为 ChineseAnalyzer() ，最后在 settings.py 中设置对应路径。Whoosh 自动建立分词和倒排索引，存放在 whoosh_index 目录下。

3.3 后端设计

我们在后端设计了如下 url 和视图函数，对前端的请求进行响应

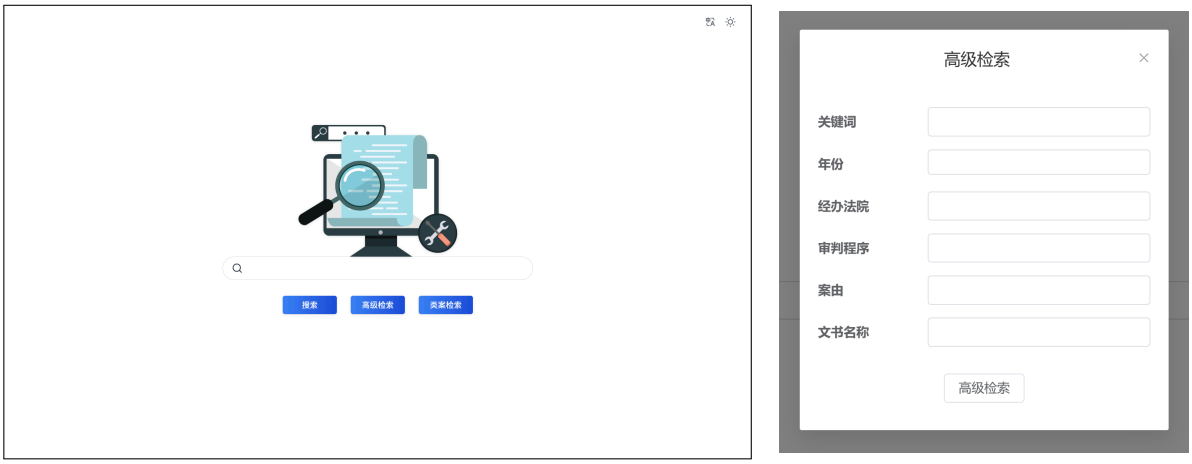
```
urlpatterns = [
    path('search/', views.search_view, name="search"),
    path("detail/", views.case_detail_view, name="case"),
    path("law2cases/", views.related_cases_from_law, name="law2cases"),
    path("upload/", views.upload_case_xml, name="upload_case_xml")
]
```

其中，`search/` 对应于全部搜索功能（关键词检索、高级检索、以案搜案、tag 搜索、法条搜索）、`detail`返回对应的案件的详细内容、`law2cases`用于返回特定法条对应的全部案件的列表、`upload`用于以案搜案功能中的文件上传。在以案搜案（上传案例文件搜索）功能的实现上，首先前端通过 POST 请求将案件文件上传之后端，存储于`/static/upload.txt`。之后前端跳转至搜索页面，在搜索页创建时前端对后端发送 GET 请求，返回搜索得到的案件列表。由于前端不同页面之间的数据传输较为不便，搜索列表和案件详情的数据获取都是在列表页和详情页的创建过程中进行的，在页面的`onMounted`或`onUpdate`函数被调用时前端向后端发送 GET 请求。

3.4 前端页面

前端页面共有三个，入口首页面、查询页面和详情页。

入口首页和查询页面均支持基础搜索、高级检索和类案检索三种搜索功能。



(a) 搜索页面　　**(b)** 高级搜索对话框

图 1: 搜索引擎首页面

搜索列表页左侧为搜索结果，上方展示了搜索条数和用时。每页展示 10 条案件的题目、摘要和关键词信息，点击题目可查看案件详情，点击 tag 则对于该关键词的显示内容进行过滤，得到对应的搜索列表。右侧同样支持高级检索和以案搜案功能。页面实现了关键词的高亮显示。

图 2: 搜索引擎查询页面

详情页面展示案件的题目，关键词信息，并按照当事人、诉讼记录、案件基本情况、判决分

析过程和判决结果五部分分段展示了全文信息，最下方展示了文尾。右侧给出了相关法律条文和案例推荐，点击法条 tag 可返回该法条对应案件的搜索列表，点击推荐案例可跳转到该案件的详情页。

图 3: 搜索引擎详情页面

4 功能实现

4.1 基础搜索

基础搜索支持对于给定搜索词句，返回相关的案例。用户可在入口首页面或者查询页面输入需要搜索的词句，随后搜索引擎将采用 Whoosh 框架自动对搜索词句进行分词去除停用词等处理，对于符合要求的返回结果进行随机排序返回，返回的结果采用分页器支持翻页查看。

在每次搜索中，我们最多展示前 100 条结果。对于每一条返回结果，我们展示了案件的标题（文首）、摘要和案件关键词（彩色 tag）。我们支持搜索关键词的高亮显示（红色），并通过后端的处理使得摘要中尽量包含对应的高亮内容。对每个案件，我们也通过标签（tag）展示其年份、案由、审判程序和文书名称；用户点击 tag 可搜索到相同类别的案件列表。

4.2 案件详情显示

用户点击感兴趣的法律案件标题即可进入详情页，详情页分为左右两部分，左侧包含案件全文和案件关键标签的汇总展示（包含年份、经办法院、审判程序、文书名称和案由），右侧为

案件涉及的法律条文一览，以及相关案例的推荐。相关案例的推荐主要参考了案由、审判程序等字段，返回与当前显示的案件类别相同的案件列表，并展示前 6 条。

4.3 高级检索

高级检索支持关键词、年份、经办法院、审判程序、案由、文书名称六个字段的共同匹配。其中，关键词字段为模糊匹配，其余字段均为精确匹配。用户可以在查询页面使用高级搜索，选取一个或多个字段进行案件搜索，搜索引擎将返回各个字段搜索结果的交集。

4.4 类案检索

类案检索支持上传案件并检索相似案件。

用户可以任意文件格式上传法律案件，如果文件为 xml 文件且与原始数据格式相同，包含审判程序、案由、文书名称等字段，搜索引擎则直接返回上述字段相同的案件列表。对于不满足格式要求的 xml 文件或其余类型文件，我们从全文中搜索出一些核心的关键词，使用这些关键词查询相关的案件。关键词设计和检索规则如下：

```
COMMON_WORDS = [
    ["交通肇事","故意伤害","强奸","非法拘禁","抢劫","盗窃","诈骗","抢夺","职务侵占"
        ,"敲诈勒索","妨害公务","聚众斗殴","寻衅滋事","走私","毒品","故意杀人","伤害"
        ],
    ["再审","调解",'仲裁'],
    ["有限公司", "合同纠纷", "财产","违约","合同"],
    ["著作权","名誉","肖像","影视作品","传播","微信","版权"],
    ["借贷","股票","报酬","工资",'信用卡',"租赁","资产","劳动"],
    ["无期徒刑", "死刑"]
]
```

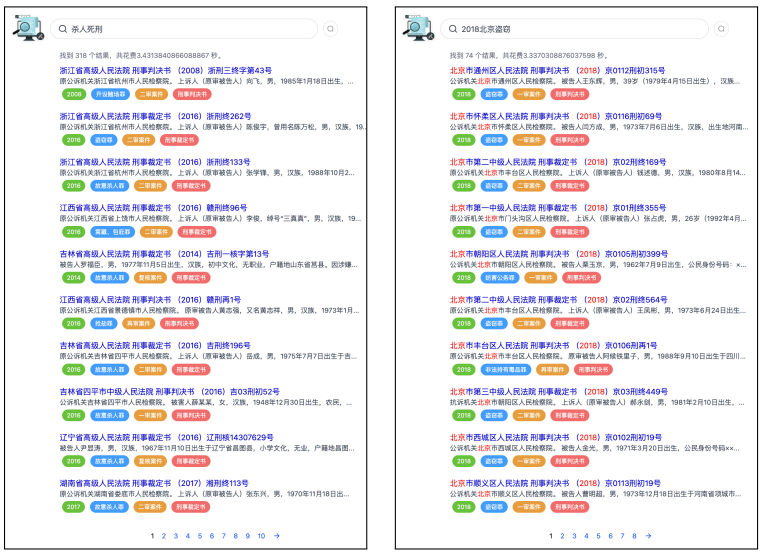
上述关键词分别对应犯罪种类、特殊审判程序、公司有关案件、创作与版权、个人劳动与收获、重大案件，从不同维度对案件的特征进行了描述。我们从全文中查找上述关键词，每组中选取不超过两个，总计不超过 5 个构建查询词集合，之后使用关键词搜索方式（切词 + 模糊查询）搜索得到类似案件。尽管上述方法实现较为朴素，但在测试中对于大多数案件取得了较好的效果。

5 测试结果及样例分析

5.1 关键词测试

查询样例	耗时	结果 1	结果 2	结果 3	结果 4	结果 5	结果 6	结果 7	结果 8	结果 9	结果 10
杀人死刑	3.533s	相关	相关	相关	相关	相关	相关	相关	相关	相关	相关
2018 北京盗窃	3.740s	相关	相关	相关	相关	不相关	相关	不相关	相关	相关	相关

表 1: 关键词测试



(a) “杀人死刑”返回结果　　**(b)** “2018 北京盗窃”返回结果

图 4: 关键词搜索测试

5.2 案例测试

查询样例	耗时	结果 1	结果 2	结果 3	结果 4	结果 5	结果 6	结果 7	结果 8	结果 9	结果 10
86866.xml	9.092s	相关									
江歌杀人案.txt	3.745s	不相关	不相关	不相关	相关	不相关	不相关	不相关	不相关	相关	相关
湖南省桂阳县人民检察院.txt	4.556s	相关	不相关	相关	相关	不相关	不相关	相关	不相关	相关	相关

表 2: 案例测试

(a) 86866.xml　　**(b)** 江歌杀人案.txt　　**(c)** 湖南省桂阳县人民检察院.txt

图 5: 案例搜索测试

纵上，搜索引擎的平均准确率（AP）为 0.79、Precision@10（P@10）为 0.74、Reciprocal Rank（RR）为 0.85、Success@10（success@10）为 1。

6 额外说明

在当面检查时，助教反应我们基于 django+whoosh 的实现较慢。我们在调试过程中也发现了类似的问题，目前没有找到问题具体所在和解决方式，在此进行记录。

```
# ./backend/cases/search_index.py

# 实现方式一
from whoosh.qparser import QueryParser
index_path = 'whoosh_index'
index = open_dir(index_path)
parser = QueryParser("text", index.schema)
query = parser.parse(query_str)
with index.searcher() as searcher:
    results = searcher.search(query, limit=None)

# 实现方式二
from haystack.forms import SearchForm, SearchQuerySet
results = SearchQuerySet().models(Case).filter(qw_value = cd['q']).load_all()
```

其中，第一种方式直接使用了 whoosh 库，而第二种方式使用了 haystack 这一对接框架。第二种方式得到的查询结果可以继续使用 filter 函数，通过其余的查询条件进行过滤（and 操作），从而进行多条件的查询。我们使用“北京盗窃”作为查询的关键词，使用上述两种方式分别获取案件列表。此外，我们还可以直接利用 django 的 model 模型自带的 filter 函数进行查询，我们以案由=故意伤害罪为例作为第三种方式，对上述方式进行实验，结果如下表所示：

查询方式	返回类型	总条数	总时间	解析 URL	过滤	计算总数	切片	分页	整理为字典
whoosh	dict	320	2.118s	0.001s	2.115s	0.000s	0.001s	0.000s	0.001s
haystack	haystack.models.SearchResult	320	8.053s	0.001s	0.522s	3.705s	3.823s	0.000s	0.002s
model	Case	744	0.846s	0.002s	0.001s	0.839s	0.002s	0.000s	0.002s

表 3: 不同查询方式的时间对比

其中解析 URL 用的是`haystack.form.SearchForm`类完成。此外，如果在这里先取 100 条再使用`len`计算数量，则使用第三种方式，利用 django 自带的 model 查询的时间可以缩短到 0.1s 左右。可以看到，主要的问题并非在于搜索过程，而是在于对模型的长度计算和切片上消耗的时间过长，其中 Haystack 库相较于 django 自带的 model 这一点更为明显。我上 google 上搜到了类似的情况，但没有找到合适的解决方案。这一问题可留待未来解决。

针对于上述问题，我们不得已需要设计如下逻辑：若只有关键词进行搜索，则直接使用 whoosh 库；若不包含关键词，则使用 django 自带的 model 模型；只有在既包含关键词也包含其它字段的时候，我们不得已才使用 haystack。

7 参考资料

在作业的实现过程中，我们主要参考了 django、Vue、Whoosh、ElementUI 的说明文档，并对于具体问题在参考了 CSDN 和 stackoverflow 上的解决方案。在作业的开始阶段，我们学习了网络上一份 Django + Haystack + Whoosh 实现的搜索引擎代码，了解了搜索引擎的大致实现方式。

8 总结

在本次作业中，我们完成了一个基于 django + Vue 的司法搜索引擎，可以通过关键词、高级搜索、以案搜案、关键词与法条查找等多种方式得到相关的案件列表，并查看案件的具体信息。在本次法律搜索引擎的大作业中，我们共同研究和实践了不同搜索框架的使用，我们一起分析需求、制定计划，并协作完成了法律搜索系统的搭建和调优。通过大作业，我们巩固了课上所学知识，深化了索引构建、查询解析、性能优化等方面的实际操作技能，同时也锻炼了团队合作和沟通能力。

8.1 小组分工

- **熊天翼**：主要负责基础搜索、高级搜索、相似案件推荐的功能实现，实现了对应的前端页面和后端响应函数，并完成了前后端的对接。
- **严澜**：主要负责数据处理与导入、whoosh 框架的引入、以案搜案功能，共同完成了部分前后端的调试与测试。

我们共同完成了前期调研、框架整体设计，以及后期的代码调试和文档撰写。

8.2 结语

感谢刘老师和艾老师的辛勤付出和专业教导，以及两位助教学长的耐心答疑解惑，在本学期的搜索引擎课程中，我们对搜索引擎有了全面的认识，不但了解到了各个子系统的原理和技术，也在业界的专家讲座中管窥了推广搜的实战。感谢助教学长们，在课程的进行以及作业完成过程中对我们提供细致和耐心的指导。

作为毕业年级的学生，可以在本科生活的最后与搜索引擎这门课相遇也是一种奇妙的缘分。回顾过去，在贵系上过的课、完成的作业以及遇到的人都会成为我们人生中的宝贵回忆。我们欣喜地看到，包括搜索引擎在内的很多课程在讲授内容和作业布置方面能够与时俱进，提升课程的意义以及同学们的学习效率，并通过午餐会等方式积极获取学生的反馈。相信在老师、助教和同学们的不断努力下，我们一定能向成为世界一流大学和院系的目标不断前进。