信息检索系统

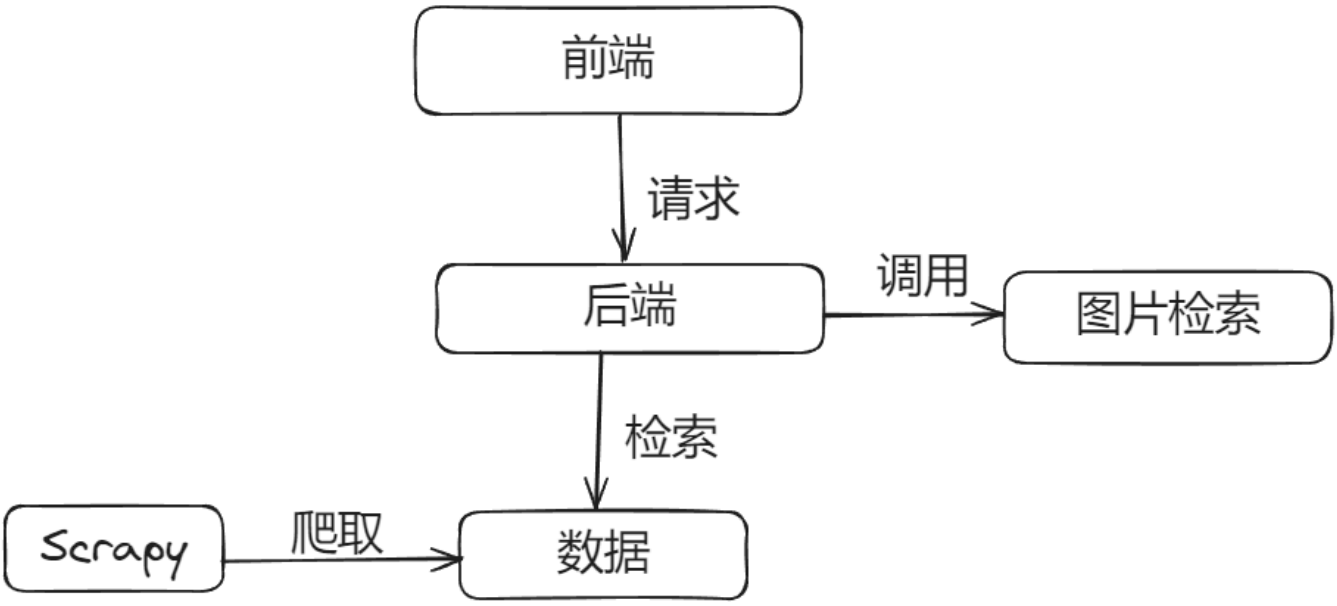
项目概述

项目要求

本实验要求自己动手设计实现一个信息检索系统，中、英文皆可，数据源可以自选，数据通过开源的网络爬虫获取，规模不低于 100 篇文档，进行本地存储。中文可以分词（可用开源代码），也可以不分词，直接使用字作为基本单元。英文可以直接通过空格分隔。构建基本的倒排索引文件。实现基本的向量空间检索模型的匹配算法。用户查询输入可以是自然语言字串，查询结果输出按相关度从大到小排序，列出相关度、题目、主要匹配内容、URL、日期等信息。最好能对检索结果的准确率进行人工评价。界面不做强制要求，可以是命令行，也可以是可操作的界面。提交作业报告和源代码。

项目架构

本项目主要由四个部分组成：数据爬取、前端展示、后端处理数据和图片检索服务，整体架构图和详细内容如下：



- 数据爬取使用 Scrapy 框架爬取网页文章并存储到 json 文件中，方便后续使用；
- 前端展示使用 Vue 框架实现，为用户提供了清晰直观的操作界面；
- 后端使用基于 Go 的 Gin 框架开发，接收用户请求并处理，并且对文章进行初始化处理，如：使用结巴框架进行分词、建立倒排索引、计算 TF-IDF 值等；
- 图片检索服务则使用 Python 进行开发，使用 Flask 框架给后端提供调用接口，使用 Tensorflow 框架对图片进行识别并提取关键词用于检索。

项目运行截图如下：

前端界面展示：

信息与知识获取

Knowledge Acquisition

关键词搜索 输入关键字 rust 搜索

图片检索 点击上传图片 搜索

Unsafe Rust

2024-05-31

介绍：All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile ...
相关度：166.90490963156572
链接： https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html

☆ ☆ ☆ ☆ ☆

信息与知识获取

图片检索 点击上传图片 搜索

Unsafe Rust
2024-05-31

介绍：All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile ...
相关度：166.90490963156572
链接： https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html

☆ ☆ ☆ ☆ ☆

Appendix D - Useful Development Tools

2024-05-31

介绍：In this appendix, we talk about some useful development tools that the Rust project provides. We'l...
相关度：0.44010686821161077
链接： https://doc.rust-lang.org/book/appendix-04-useful-development-tools.html

☆ ☆ ☆ ☆ ☆

RefCell<T> and the Interior Mutability Pattern

2024-05-31

介绍： is a design pattern in Rust that allows you to mutate data even when there are immutable references...
相关度：0.0784623650376175
链接： https://doc.rust-lang.org/book/ch15-05-interior-mutability.html

☆ ☆ ☆ ☆ ☆

Unsafe Rust

2024-05-31

介绍：All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile ...
相关度：166.90490963156572%
链接： https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html

Unsafe Rust

All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile time. However, Rust has a second language hidden inside it that doesn't enforce these memory safety guarantees: it's called *unsafe Rust* and works just like regular Rust, but gives us extra superpowers.

Unsafe Rust exists because, by nature, static analysis is conservative. When the compiler tries to determine whether or not code upholds the guarantees, it's better for it to reject some valid programs than to accept some invalid programs. Although the code *might* be okay, if the Rust compiler doesn't have enough information to be confident, it will reject the code. In these cases, you can use unsafe code to tell the compiler, "Trust me, I know what I'm doing." Be warned, however, that you use unsafe Rust at your own risk: if you use unsafe code incorrectly, problems can occur due to memory unsafety, such as null pointer dereferencing.

Another reason Rust has an unsafe alter ego is that the underlying computer hardware is inherently unsafe. If Rust didn't let you do unsafe operations, you couldn't do certain tasks. Rust needs to allow you to do low-level systems programming, such as directly interacting with the operating system or even writing your own operating system. Working with low-level systems programming is one of the goals of the language. Let's explore what we can do with unsafe Rust and how to do it.

Unsafe Superpowers

To switch to unsafe Rust, use the `unsafe` keyword and then start a new block that holds the unsafe code. You can take five actions in unsafe Rust that you can't in safe Rust, which we call *unsafe superpowers*. Those superpowers include the ability to:

- Dereference a raw pointer
- Call an unsafe function or method
- Access or modify a mutable static variable
- Implement an unsafe trait
- Access fields of `union`s

It's important to understand that `unsafe` doesn't turn off the borrow checker or disable any other of Rust's safety checks: if you use a reference in unsafe code, it will still be checked. The `unsafe` keyword only gives you access to these five features that are then not checked by the compiler for memory safety. You'll still get some degree of safety inside of an unsafe block.

In addition, `unsafe` does not mean the code inside the block is necessarily dangerous or that it will definitely have memory safety problems: the intent is that as the programmer, you'll

信息与知识获取

Accessing Fields of a Union

The final action that works only with `unsafe` is accessing fields of a *union*. A `union` is similar to a `struct`, but only one declared field is used in a particular instance at one time. Unions are primarily used to interface with unions in C code. Accessing union fields is unsafe because Rust can't guarantee the type of the data currently being stored in the union instance. You can learn more about unions in the Rust Reference.

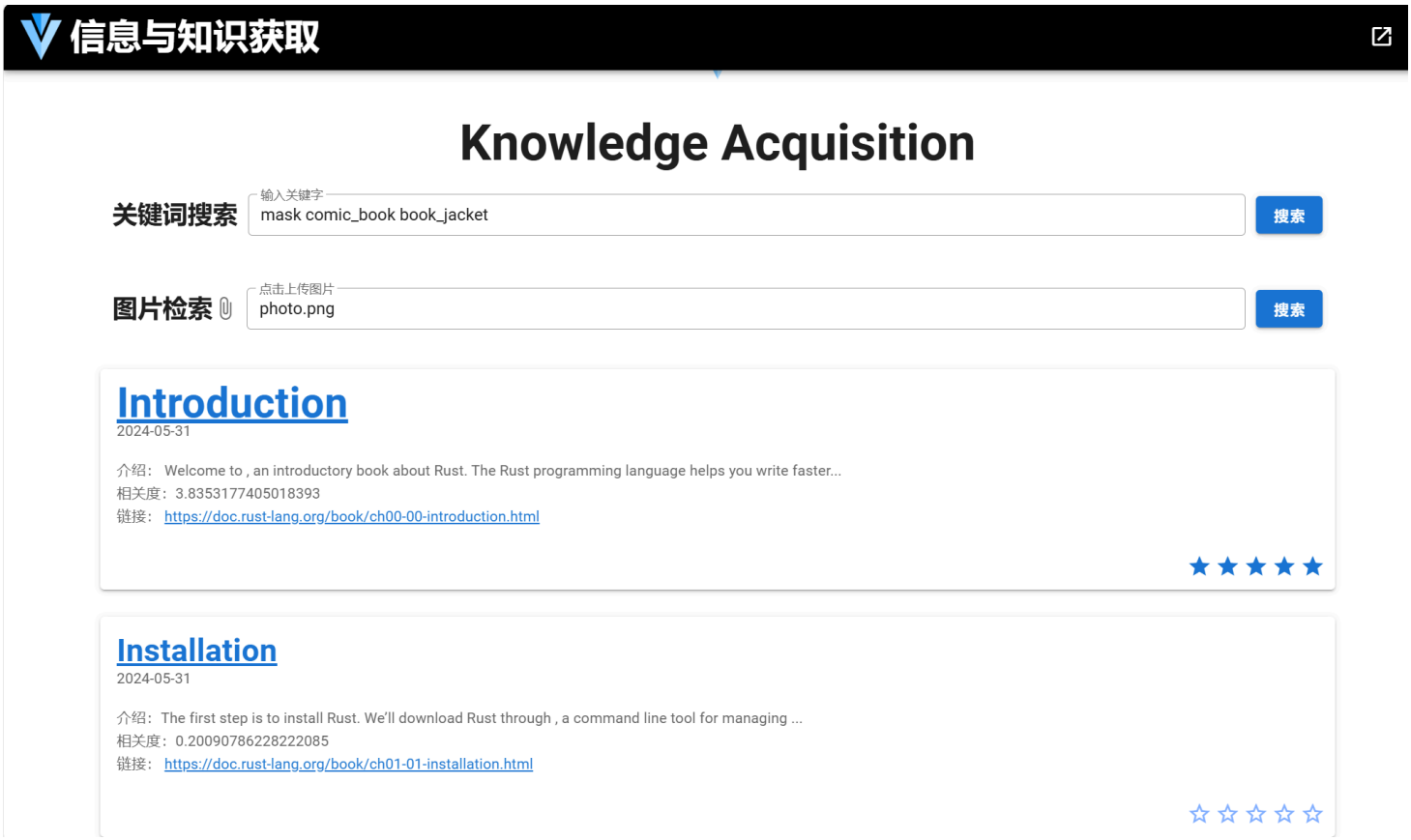
When to Use Unsafe Code

Using `unsafe` to take one of the five actions (superpowers) just discussed isn't wrong or even frowned upon. But it is trickier to get `unsafe` code correct because the compiler can't help uphold memory safety. When you have a reason to use `unsafe` code, you can do so, and having the explicit `unsafe` annotation makes it easier to track down the source of problems when they occur.

关键字: All the code we've discussed so far has had Rust's...
语言: 英文

实体	频率	反馈
ABI	3	☆ ☆ ☆ ☆ ☆
Listing	3	☆ ☆ ☆ ☆ ☆
five	5	☆ ☆ ☆ ☆ ☆
one	4	☆ ☆ ☆ ☆ ☆
two	5	☆ ☆ ☆ ☆ ☆
热词	**频率**	**反馈**
Rust	36	☆ ☆ ☆ ☆ ☆
code	41	☆ ☆ ☆ ☆ ☆
raw	27	☆ ☆ ☆ ☆ ☆
unsafe	45	☆ ☆ ☆ ☆ ☆
'	95	☆ ☆ ☆ ☆ ☆

☆ ☆ ☆ ☆ ☆



后端日志展示：

```
[GIN] 2024/06/08 - 21:45:55 | 200 |          606µs |     10.29.94.205 | GET      "/api/v1/search?q=test&page=1&limit=10"
[GIN] 2024/06/08 - 21:46:40 | 200 |      1.0925ms |     10.29.94.205 | GET      "/api/v1/search?q=test&page=1&limit=10"
time="2024-06-08T21:50:30+08:00" level=info msg="queryWords: [1]"
time="2024-06-08T21:50:30+08:00" level=info msg="queryVector:map[1:-0.9999983468443772]"
time="2024-06-08T21:50:30+08:00" level=info msg="202 results"
time="2024-06-08T21:50:30+08:00" level=debug msg=">>> scoreMap"
time="2024-06-08T21:50:30+08:00" level=debug msg="83:Doc:{83 <h2 id=\"storing-keys-with-associated-values-in-hash-maps\"><a class=\"header\" href=\"#storing-keys-with-associated-values-in-hash-maps\">Storing Keys with Associated Values in Hash Maps</a></h2> The last of our common collections is the . The type \nstores a mapping of keys of type  to values of... https://doc.rust-lang.org/book/ch08-03-hash-maps.html 2024-05-31}Score:0.0001931839082104849"
time="2024-06-08T21:50:30+08:00" level=debug msg="188:Doc:{188 <h1>杨氏矩阵</h1> 杨氏矩阵引入(Young tableau)，又名杨表，是一种常用于表示论和舒伯特演算中\xe7... https://oi-wiki.org/math/young-tableau/ 2024-05-31}Score:3.9090714015602266e-05"
time="2024-06-08T21:50:30+08:00" level=debug msg="198:Doc:{198 <h1>公平组合游戏</h1> 公平组合游戏经典的公平组合游戏有很多，包括取数游戏，31 点，以及 Nim \xe6\xb8... https://oi-wiki.org/math/game-theory/impartial-game/ 2024-05-31}Score:0.0019040737123493896"
time="2024-06-08T21:50:30+08:00" level=debug msg="48:Doc:{48 <h2 id=\"using-threads-to-run-code-simultaneously\"><a class=\"header\" href=\"#using-threads-to-run-code-simultaneously\">Using Threads to Run Code Simultaneously</a></h2> In most current operating systems, an executed program's code is  run in a\n, and the operating syst... https://doc.rust-lang.org/book/ch16-01-threads.html 2024-05-31}Score:0.0004254136865664769"
time="2024-06-08T21:50:30+08:00" level=debug msg="388:Doc:{388 <h1>bitset</h1> bitset介绍 是标准库中的一个存储  的大小不可变容器。严格来讲，它并不属\xe4... https://oi-wiki.org/lang/csl/bitset/ 2024-05-31}Score:5.162520772654723e-05"
time="2024-06-08T21:50:30+08:00" level=debug msg="422:Doc:{422 <h1>出题</h1> 出题出题前的准备具备一定的水平一方面，一个人自己出题，很难出出难度\xe5... https://oi-wiki.org/contest/problemsetting/ 2024-05-31}Score:8.88146634694935e-06"
time="2024-06-08T21:50:30+08:00" level=debug msg="55:Doc:{55 <h2 id=\"cargo-workspaces\"><a class=\"header\" href=\"#cargo-workspaces\">Cargo Workspaces</a></h2> In Chapter 12, we built a package that included a binary crate and a library\ncrate. As your project ... https://doc.rust-lang.org/book/ch14-03-cargo-workspaces.html 2024-05-31}Score:6.963541137311594e-06"
time="2024-06-08T21:50:30+08:00" level=debug msg="119:Doc:{119 <h1>矩阵树定理</h1> 矩阵树定理Kirchhoff 矩阵树定理（简称矩阵树定理）解决了一张图的生成树个... https://oi-wiki.org/graph/matrix-tree/ 2024-05-31}Score:0.010477520794562186"
time="2024-06-08T21:50:30+08:00" level=debug msg="291:Doc:{291 <h1>Lyndon 分解</h1> Lyndon 分解定义首先我们介绍 Lyndon 分解的概念。Lyndon 串：对于字符串 ，如\xe6\x9e... https://oi-wiki.org/string/lyndon/ 2024-05-31}Score:6.930629463264774e-06"
time="2024-06-08T21:50:30+08:00" level=debug msg="256:Doc:{256 <h1>洲阁筛</h1> 洲阁筛前置知识定义洲阁筛是一种能在亚线性时间复杂度内求出大多数积性\xe5... https://oi-wiki.org/math/number-theory/zhou/ 2024-05-31}Score:0.0005535866616791869"
time="2024-06-08T21:50:30+08:00" level=debug msg="177:Doc:{177 <h1>划分树</h1> 划分树引入划分树是一种来解决区间第  大的一种数据结构，其常数、理解\xe9\x9a... https://oi-wiki.org/ds/dividing/ 2024-05-31}Score:6.444740970351023e-06"
time="2024-06-08T21:50:30+08:00" level=debug msg="69:Doc:{69 <h2 id=\"refactoring-to-improve-modularity-and-error-handling\"><a class=\"header\" href
```

```
time="2024-06-08T22:11:06+08:00" level=debug msg="102:Doc:{102 <h2 id=\"installation\"><a class=\"header\" href=\"#installation\">Installation</a></h
2> The first step is to install Rust. We'll download Rust through , a\ncommand line tool for managing ... https://doc.rust-lang.org/book/ch01-01-inst
allation.html 2024-05-31}Score:0.20090786228222085"
time="2024-06-08T22:11:06+08:00" level=debug msg="41:Doc:{41 <h2 id=\"extensible-concurrency-with-the-sync-and-send-traits\"><a class=\"header\" href
=\"#extensible-concurrency-with-the-sync-and-send-traits\">Extensible Concurrency with the <code>Sync</code> and <code>Send</code> Traits</a></h2> In
terestingly, the Rust language has  few concurrency features. Almost\nevery concurrency feature we\xe2... https://doc.rust-lang.org/book/ch16-04-exte
nsible-concurrency-sync-and-send.html 2024-05-31}Score:0.006258576034104325"
time="2024-06-08T22:11:06+08:00" level=debug msg="43:Doc:{43 <h2 id=\"characteristics-of-object-oriented-languages\"><a class=\"header\" href=\"#char
acteristics-of-object-oriented-languages\">Characteristics of Object-Oriented Languages</a></h2> There is no consensus in the programming community a
bout what features a\nlanguage must have to be co... https://doc.rust-lang.org/book/ch17-01-what-is-oo.html 2024-05-31}Score:0.029741694042025146"
time="2024-06-08T22:11:06+08:00" level=debug msg="<<< scoreMap"
[GIN] 2024/06/08 - 22:11:06 | 200 |     291.097ms |             ::1 | POST     "/api/v1/search_by_image"
[GIN] 2024/06/08 - 22:11:12 | 200 |       652.3µs |    10.29.94.205 | GET      "/api/v1/search?q=test&page=1&limit=10"
[GIN] 2024/06/08 - 22:11:17 | 200 |       880.5µs |    10.29.94.205 | GET      "/api/v1/document?id=78"
time="2024-06-08T22:11:17+08:00" level=debug msg="Extract info for doc 78 entities: map[8:3 Chapter 5:3 one:3 two:6 's:3] hot_words: map[code:24 func
tion:36 test:89 tests:35 ':63]"
[GIN] 2024/06/08 - 22:11:17 | 200 |    475.6573ms |    10.29.94.205 | GET      "/api/v1/extract_info?id=78"
[GIN] 2024/06/08 - 22:11:41 | 200 |      1.4874ms |    10.29.94.205 | GET      "/api/v1/document?id=65"
time="2024-06-08T22:11:42+08:00" level=debug msg="Extract info for doc 65 entities: map[Listing:3 TDD:3 one:2] hot_words: map[We:9 function:19 return
:15 test:19 ':43]"
[GIN] 2024/06/08 - 22:11:42 | 200 |    256.7817ms |    10.29.94.205 | GET      "/api/v1/extract_info?id=65"
[GIN] 2024/06/08 - 22:11:59 | 200 |       520.5µs |    10.29.94.205 | GET      "/api/v1/search?q=test&page=1&limit=10"
[GIN] 2024/06/08 - 22:12:05 | 200 |       261.7µs |    10.29.94.205 | GET      "/api/v1/document?id=69"
time="2024-06-08T22:12:05+08:00" level=debug msg="Extract info for doc 69 entities: map[1:3 12:4 Listing:6 one:5 two:5] hot_words: map[code:29 error:
29 function:51 value:29 ':95]"
[GIN] 2024/06/08 - 22:12:05 | 200 |    527.8517ms |    10.29.94.205 | GET      "/api/v1/extract_info?id=69"
```

```
10.29.12.98 - - [08/Jun/2024 22:10:06] "POST /extract_info HTTP/1.1" 200 -
[2024-06-08 22:11:07,734] INFO in main: photo.png
1/1 [==============================] - 0s 27ms/step
10.29.12.98 - - [08/Jun/2024 22:11:07] "POST /image_to_keywords HTTP/1.1" 200 -
[2024-06-08 22:11:18,740] INFO in main: Data language: en
entities: [{'text': 'three', 'label': 'CARDINAL'}, {'text': 'Rust', 'label': 'GPE'}, {'text': 'one', 'label': 'CARDINAL'}, {'
text': 'Chapter 5', 'label': 'LAW'}, {'text': 'Cargo', 'label': 'ORG'}, {'text': '’s', 'label': 'NORP'}, {'text': 'two', 'lab
el': 'CARDINAL'}, {'text': 'Listing 11-1.For', 'label': 'FAC'}, {'text': 'two', 'label': 'CARDINAL'}, {'text': '2', 'label':
'CARDINAL'}, {'text': '4', 'label': 'CARDINAL'}, {'text': 'Listing', 'label': 'GPE'}, {'text': '11-2.Cargo', 'label': 'CARDIN
AL'}, {'text': 'n’t', 'label': 'GPE'}, {'text': 'Chapter 14', 'label': 'LAW'}, {'text': 'First', 'label': 'ORDINAL'}, {'text'
: 'Chapter 9', 'label': 'LAW'}, {'text': 'Listing 11-3.Run', 'label': 'WORK_OF_ART'}, {'text': '11', 'label': 'CARDINAL'}, {'
text': 'Two', 'label': 'CARDINAL'}, {'text': 'first', 'label': 'ORDINAL'}, {'text': '10', 'label': 'CARDINAL'}, {'text': 'one
', 'label': 'CARDINAL'}, {'text': 'one', 'label': 'CARDINAL'}, {'text': '’s', 'label': 'NORP'}, {'text': 'Boolean', 'label':
'GPE'}, {'text': 'Chapter 5', 'label': 'LAW'}, {'text': 'Listing 11-5', 'label': 'FAC'}, {'text': 'Boolean', 'label': 'GPE'},
 {'text': '11-6', 'label': 'CARDINAL'}, {'text': '8', 'label': 'CARDINAL'}, {'text': '7', 'label': 'CARDINAL'}, {'text': '5',
 'label': 'CARDINAL'}, {'text': '1.Note', 'label': 'CARDINAL'}, {'text': 'Chapter 7', 'label': 'LAW'}, {'text': 'two', 'label
': 'CARDINAL'}, {'text': 'Two', 'label': 'CARDINAL'}, {'text': '8', 'label': 'CARDINAL'}, {'text': '5', 'label': 'CARDINAL'},
 {'text': '8', 'label': 'CARDINAL'}, {'text': 'less than', 'label': 'CARDINAL'}, {'text': 'two', 'label': 'CARDINAL'}, {'text
': 'two', 'label': 'CARDINAL'}, {'text': '’s', 'label': 'NORP'}, {'text': 'two', 'label': 'CARDINAL'}, {'text': 'the day', 'l
abel': 'DATE'}, {'text': 'the week', 'label': 'DATE'}, {'text': 'Chapter 5', 'label': 'LAW'}, {'text': 'Chapter 8', 'label':
'LAW'}, {'text': 'Chapter 9', 'label': 'LAW'}, {'text': 'between 1 and 100', 'label': 'CARDINAL'}, {'text': '11-8', 'label':
'CARDINAL'}, {'text': 'greater than 100', 'label': 'CARDINAL'}, {'text': 'Listing 11-8', 'label': 'FAC'}, {'text': 'Listing 1
1-9', 'label': 'FAC'}, {'text': 'Listing 11-1', 'label': 'FAC'}]
[2024-06-08 22:11:19,194] DEBUG in main: {"entities": {"two": 6, "one": 3, "Chapter 5": 3, "\u2019s": 3, "8": 3}, "hot_words"
: {"test": 89, "\u2019": 63, "function": 36, "tests": 35, "code": 24}}
```

详细设计

数据爬取

本次实验要求不少于 100 篇文档，所以我们结合自己情况爬取了比较常用的全中文的**OiWiki**和最近在学习的全英文的**The Rust Programming Language**，最终爬取文章数量为中文 **440** 篇，英文 **104** 篇。爬虫框架选取了我们最为熟悉的 Scrapy，使用该框架可以快速爬取网页内容，并且可以方便的进行数据处理。

OiWiki 数据爬取

对于 OiWiki，我们首先爬取文章列表：

```
def parse(self, response):
    sections = response.xpath(
        "//li[@class='md-nav__item']/a[@class='md-nav__link']"
    )
    hrefs = sections.xpath("@href").getall()
    texts = sections.xpath("text()").getall()
    texts = [t.strip() for t in texts]

    for href, section in zip(hrefs, texts):
        url = response.urljoin(href)
        yield scrapy.Request(
            url=url,
            callback=self.parse_section,
            cb_kwargs={"section": section},
        )
```

然后爬取每篇文章内容：

```
def parse_section(self, response, section="Unknown"):
    content = response.xpath(
        '//div[@class="md-content"]//blockquote[1]/preceding-sibling::*[not(self::a)]'
    ).getall()
    keywords = response.xpath(
        '//div[@class="md-content"]//*[self::h1 or self::h2 or self::h3 or self::h4 or self::li
    ).getall()

    self.id = self.id + 1
    yield {
        "id": str(self.id),
        "title": content[0],
        "content": "".join(para for para in content),
        "keywords": "".join(para for para in keywords),
        "url": response.url,
        "date": datetime.date.today().strftime("%Y-%m-%d"),
    }
```

其中，我们将全文内容作为文章内容用于前端展示，文章中的所有文本内容作为关键字用于索引和检索。

The Rust Programming Language 数据爬取

对于 The Rust Programming Language，我们同样地也是先爬取文章所有章节，然后再爬取每个章节内的详细内容：

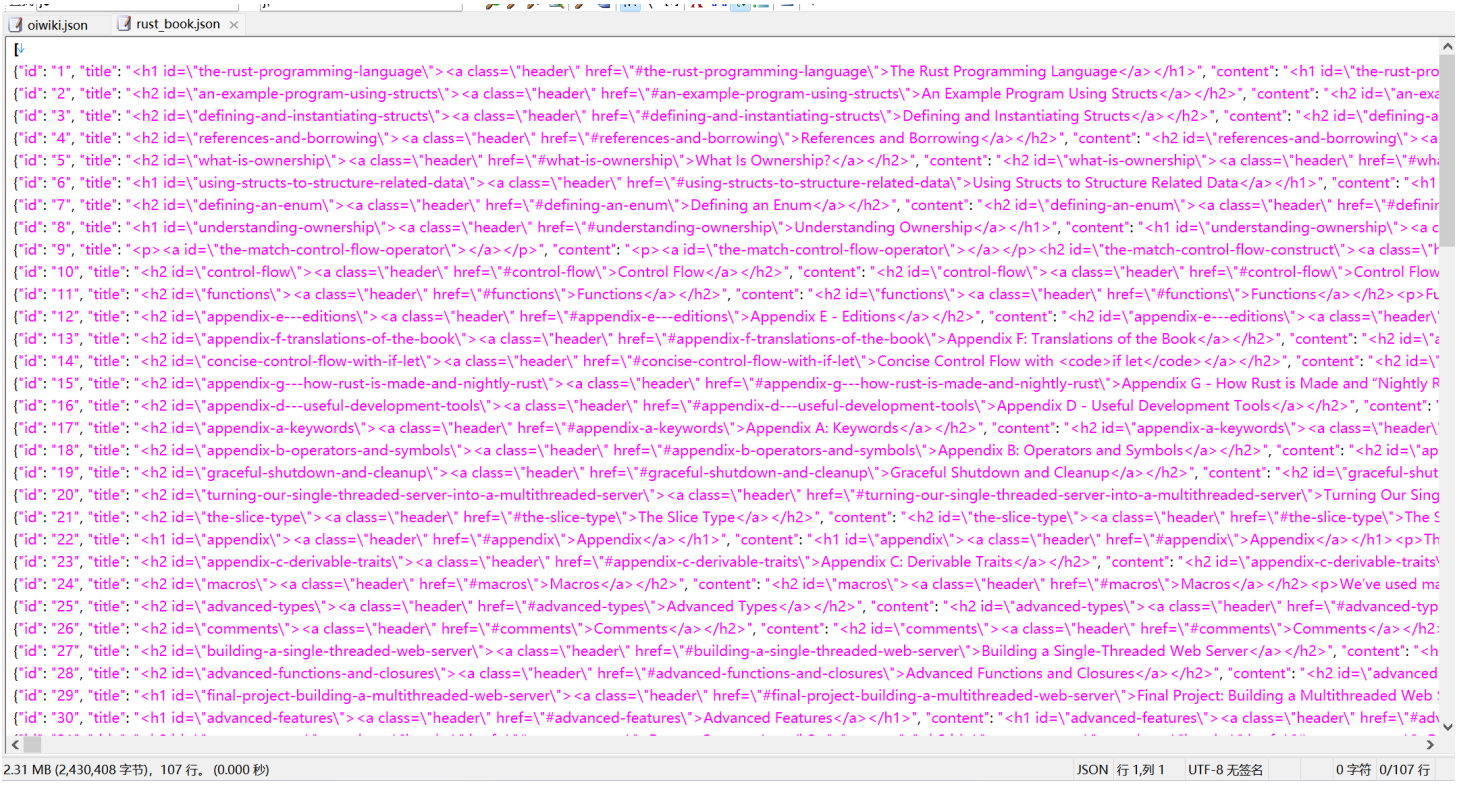
```
def parse(self, response):
    chapters = response.xpath(
        '//ol[@class="chapter"]//li[@class="chapter-item expanded " or @class="chapter-item expa
    )
    hrefs = chapters.xpath("@href").getall()
    texts = chapters.xpath("text()").getall()

    for href, chapter in zip(hrefs, texts):
        url = response.urljoin(href)
        yield scrapy.Request(
            url=url,
            callback=self.parse_chapter,
            cb_kwargs={"chapter": chapter},
        )
def parse_chapter(self, response, chapter="Unknown"):
    content = response.xpath("//main/*")
    keywords = content.xpath("text()").getall()
    content = content.getall()

    self.id = self.id + 1
    yield {
        "id": str(self.id),
        "title": content[0],
        "content": "".join(p for p in content),
        "keywords": "".join(p for p in keywords),
        "url": response.url,
        "date": datetime.date.today().strftime("%Y-%m-%d"),
    }
```

数据存储

为了方便后续索引和检索，我们将所有爬取到的数据都存储成 `json` 文件，每一条的数据的形式如下，保证所有要求的必要信息都会被存储：

```
{
    "id": "1",
    "title": "The Rust Programming Language",
    "content": "The Rust Programming Language",
    "keywords": "The Rust Programming Language",
    "url": "https://doc.rust-lang.org/book/",
    "date": "2022-04-27"
}
```


最终的爬取结果如下：



后端设计与实现

因为 Go 在各种测试中表现出了优秀的性能水平，所以本次实验后端我使用 Go 语言进行开发，框架使用了 Gin 这一高性能且比较主流的 Web 框架，本部分将介绍本项目的后端接口设计以及相关逻辑实现。

不过 Go 令人最为诟病的一点就是其 `err` 的判断机制，几乎每一次函数调用都要判断函数返回的 `err` 是否需要处理，所以考虑到报告的篇幅长度，我在后续展示 Go 代码时都将删去 `err` 的判断部分，以便带来更加良好的阅读体验。

接口展示

首先展示一下我们此次实验完成的所有接口（包括作业二和作业三）

查询接口

- GET /document 获取文档详细信息
- GET /search 分页查询
- POST /search_by_image 上传图片查询

反馈接口

- POST /entity_feedback 实体反馈
- POST /extract_info_regex_feedback 正则提取反馈 (正则+词性)
- POST /feedback 结果反馈
- POST /hotword_feedback 热词反馈

提取接口

- GET /extract_info 提取关键信息
- GET /extract_info_regex 提取关键信息

```
r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerFiles.Handler))

v1 := r.Group("/api/v1")
{
    // Search with keywords
    v1.GET("/search", controller.Search)
    // Fetch SearchResult content details
    v1.GET("/document", controller.GetDocument)
    // Search by image
    v1.POST("/search_by_image", controller.SearchByImage)

    // Entities and hot words
    v1.GET("/extract_info", controller.ExtractInfo)
    // Entity and hot word feedback
    v1.POST("/extract_info_regex", controller.ExtractInfoRegex)

    // Feedback
    v1.POST("/feedback", controller.Feedback)
    // Entity Feedback
    v1.POST("/entity_feedback", controller.EntityFeedback)
    // Hotword Feedback
    v1.POST("/hotword_feedback", controller.HotwordFeedback)
    // Regex Feedback
    v1.POST("/extract_info_regex_feedback", controller.ExtractInfoRegexFeedback)
}
```

关键词搜索

根据关键词时用户提交关键词，请求到达后端后，由 controller 调用对应的 logic 函数实现具体功能。具体实现过程是先查询缓存，缓存未命中则对关键词进行分词，然后调用核心的 `SearchIndex` 函数进行查询， `SearchIndex` 函数的具体实现将会在信息检索服务中详细介绍，查询结束后则将结果放入缓存并向前端返回查询结果。

```go
func Search(q string, page string, resultsPerPage string) (r model.SearchResponse, err error) {
    cacheKey := fmt.Sprintf("%s-%s-%s", q, page, resultsPerPage)

    if cachedResults, found := cache.Get(cacheKey); found {
        return model.SearchResponse{Code: 200, Results: cachedResults}, nil
    }

    intPage, err := strconv.Atoi(page)

    intResultsPerPage, err := strconv.Atoi(resultsPerPage)

    queryWords := WordSplit(q)
    log.Info("queryWords: ", queryWords)

    results, err := SearchIndex(queryWords, intPage, intResultsPerPage)

    cache.Set(cacheKey, results)

    return model.SearchResponse{Code: 200, Results: results}, nil
}
```

图片搜索

图片搜索需要调用 python 写的接口，通过模型识别从图片中提取关键字，然后使用关键词进行搜索。所以在 controller 中，图片搜索首先调用 `SearchByImageLogic` 函数，该函数会调用 python 接口提取图片关键词，使用 python 对图片的处理则会在多媒体关键词提取服务中详细介绍。

```go
func SearchByImageLogic(imagePath string) (string, error) {
    var b bytes.Buffer
    w := multipart.NewWriter(&b)
    f, err := os.Open(imagePath)

    defer f.Close()

    fw, err := w.CreateFormFile("file", filepath.Base(imagePath))
    if _, err = io.Copy(fw, f); err != nil {
        return "", err
    }
    w.Close()

    req, err := http.NewRequest("POST", model.PYTHON_SERVER_URL+"/image_to_keywords", &b)
    req.Header.Set("Content-Type", w.FormDataContentType())

    client := &http.Client{}
    res, err := client.Do(req)

    body, err := ioutil.ReadAll(res.Body)

    var kr KeywordResponse
    err = json.Unmarshal(body, &kr)

    return kr.Keyword, nil
}
```

获取文章内容

考虑到网络传输，每次查询结果都是返回文章概要信息，并不会返回文章的详细内容，所以我另外提供了一个根据文章 `id` 获取文章详细内容的接口，用于返回文章的概要信息和文章的具体内容。

```go
// 直接从map中查询到文章内容并返回即可
func GetFullDoc(id string) (model.Document, bool) {
    doc, ok := idDocMap[id]
    return doc, ok
}
```

前端设计与实现

框架选择与设计理念

我们选择 **Vue.js** 作为前端开发的框架，并结合 **Vuetify** 组件库，符合现代网页设计趋势，又能提供丰富的交互元素来增强用户体验。

本项目的前端界面设计旨在提供清晰、直观的用户交互体验。通过简洁的设计风格和直观的操作流程，用户能够轻松进行信息检索，无论是通过关键词搜索还是图片检索。设计重点放在用户操作的便捷性和界面的响应速度上，以适应不同背景知识的用户。

界面设计与接口实现

初始界面：

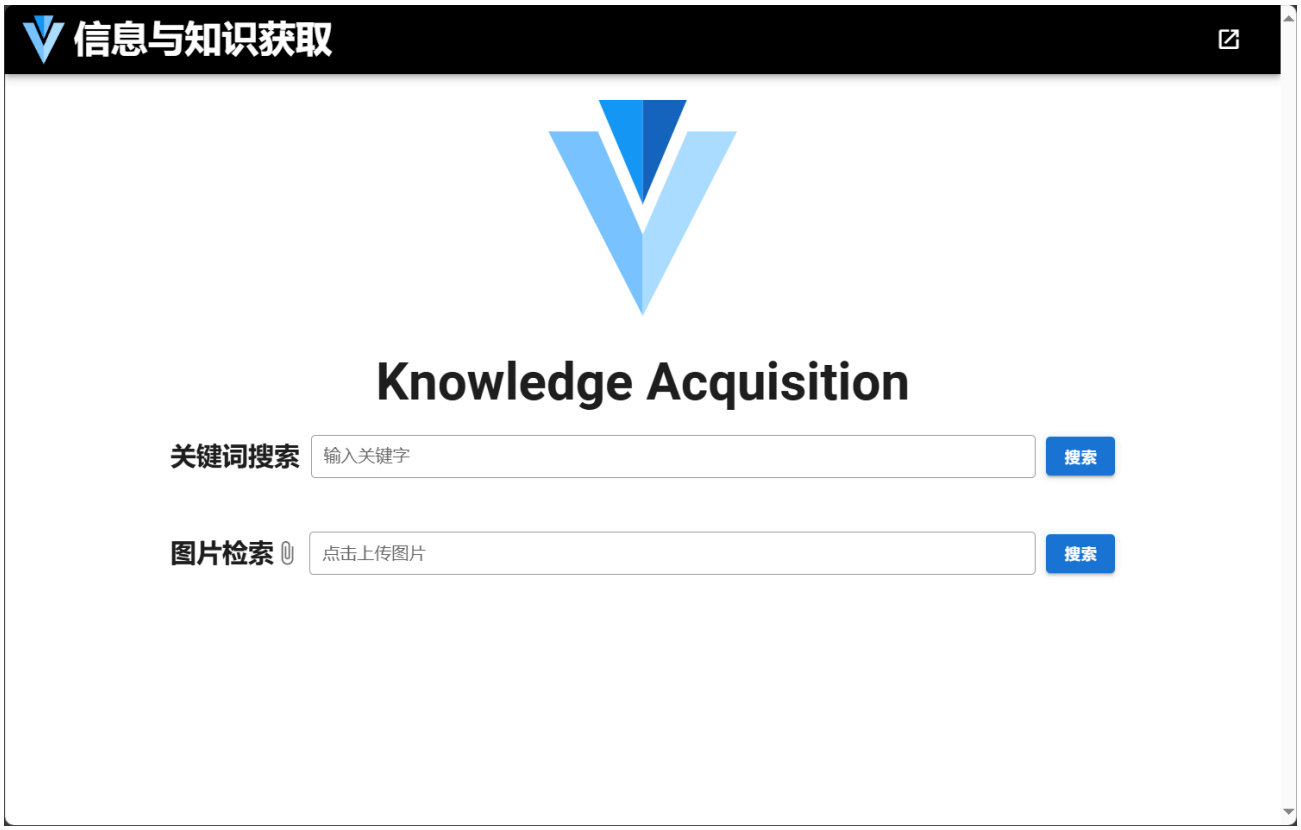


image-20240609155511735

顶栏与 Logo：

- **组件**： `<v-img>`
- **功能**：在页面顶部显示顶栏，正文顶部中央展示系统的 Logo。

Knowledge Acquisition

- image-20240609155149992

关键词搜索

- image-20240609155130296

- **组件**： `<v-text-field>` 和 `<v-btn>`
- **功能**：
  - `v-text-field`：允许用户输入搜索关键词，可以有多个关键词，关键词之间空格隔开。
  - `v-btn`：点击后触发搜索功能，按钮文本为“搜索”，颜色为主题色，增加视觉效果。
- **布局**：关键词输入框和搜索按钮在同一行显示，使用 Vuetify 的布局系统优化空间利用。
- **接口函数：**

```
searchByKeyword() {
   const params = { q: this.searchText };
   axios
     .get(`api/v1/search`, { params })
     .then((response) => {
        this.searchResults = response.data;
     })
     .catch((error) => {
        console.error("Error during keyword search:", error);
     });
},
```

图片检索

- image-20240609161432057

- **组件**： `<v-file-input>` 和 `<v-btn>`
- **功能**：
  - `v-file-input`： 提供图片上传功能，标签“点击上传图片”指导用户操作，支持文件类型过滤。
  - `v-btn`： 用于触发图片的上传和检索处理，保持与关键词搜索按钮一致的风格。
    点击搜索按钮后，后端将识别图片并返回与输入图片相关的关键词显示在关键词搜索框中。
- **布局**：与关键词搜索类似，确保操作的一致性和界面的整洁。
- **接口函数：**

```
searchByImage() {
  if (!this.imageFile) {
    alert("Please upload an image.");
    return;
  }
  const formData = new FormData();
  formData.append("image", this.imageFile);
  axios
    .post(`api/v1/search_by_image`, formData, {
      headers: { "Content-Type": "multipart/form-data" },
    })
    .then((response) => {
      this.searchResults = response.data.results;
      this.searchText = response.data.keywords;
    })
    .catch((error) => {
      console.error("Error during image search:", error);
    });
},
```

搜索结果展示

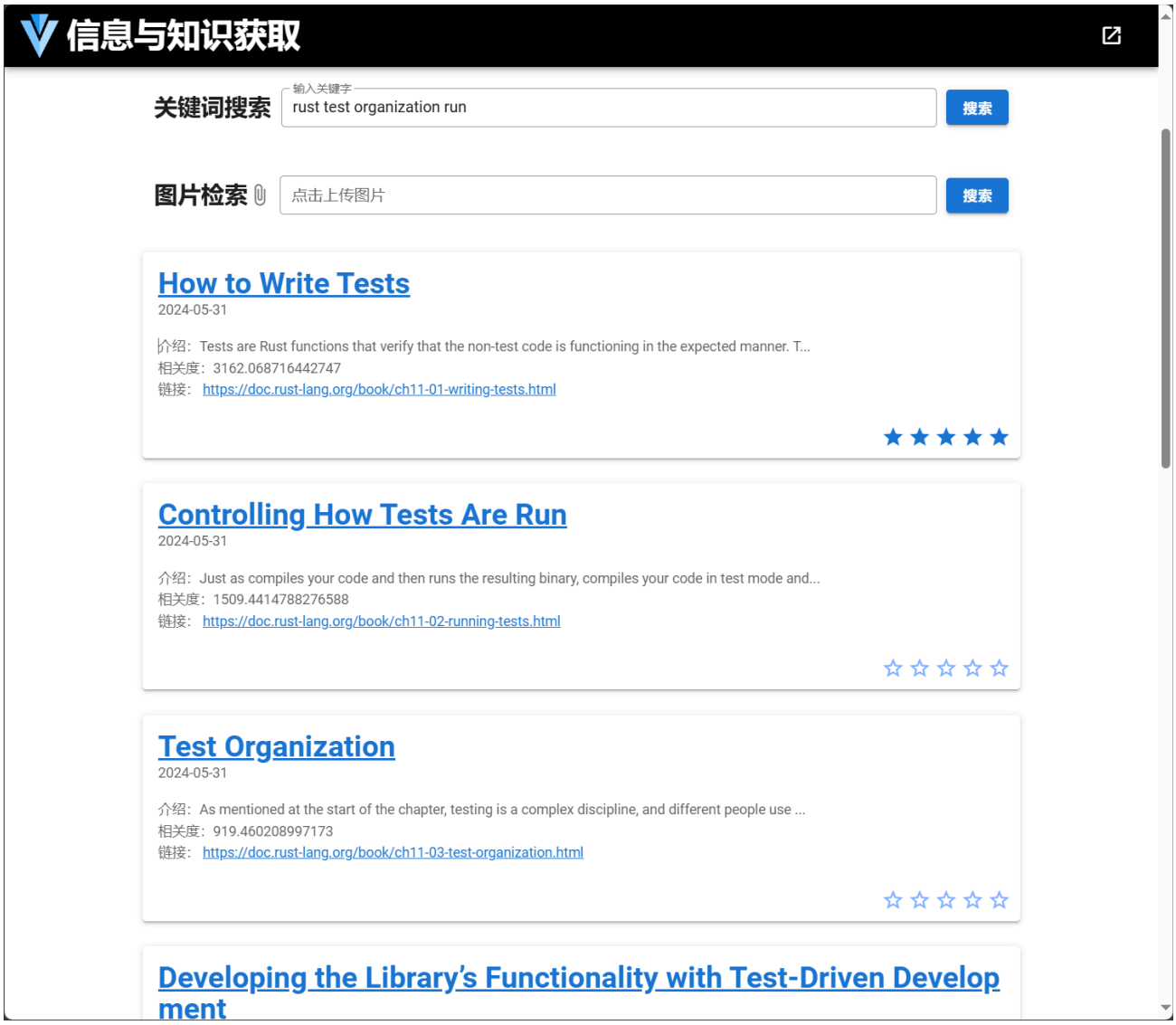
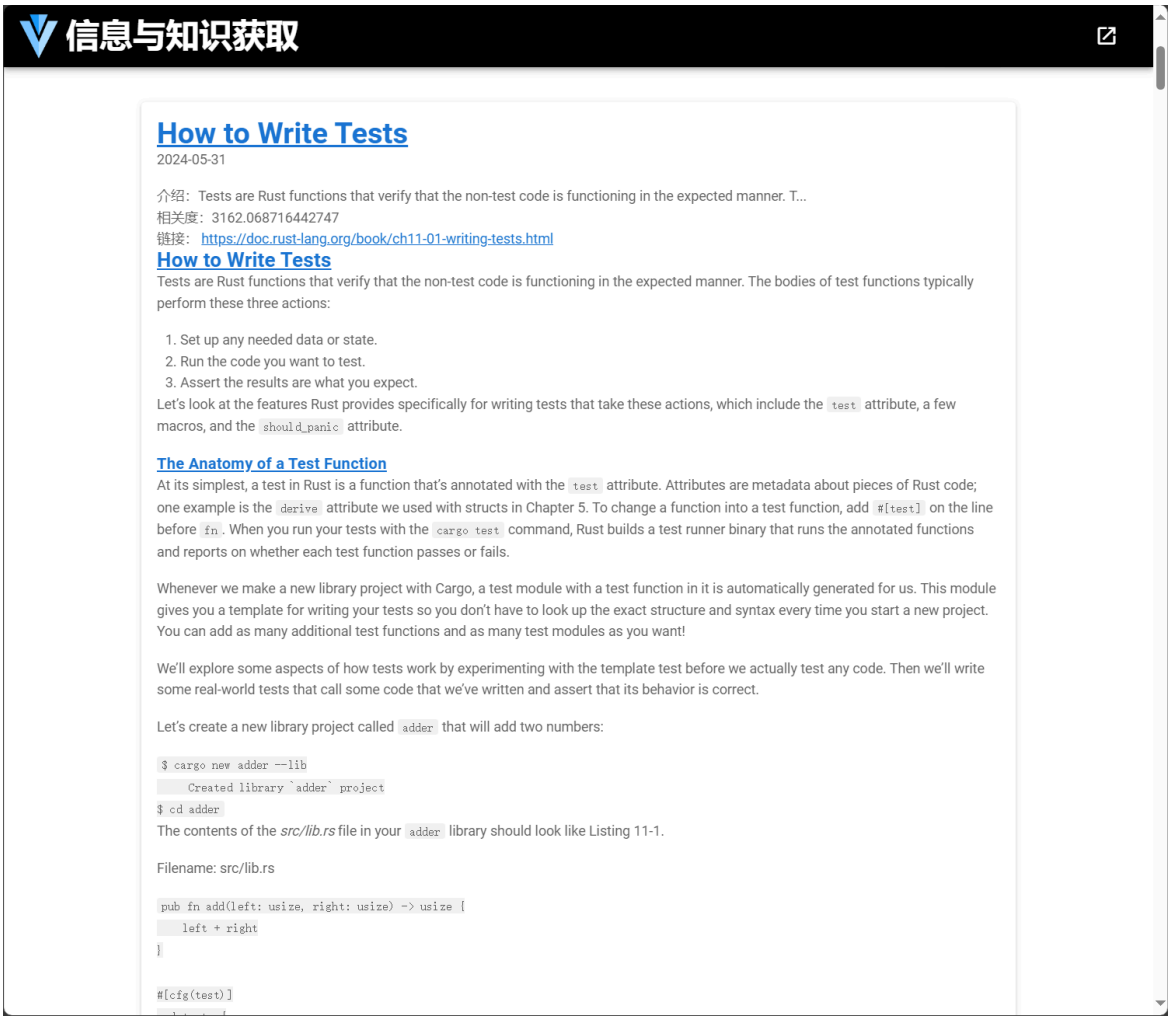
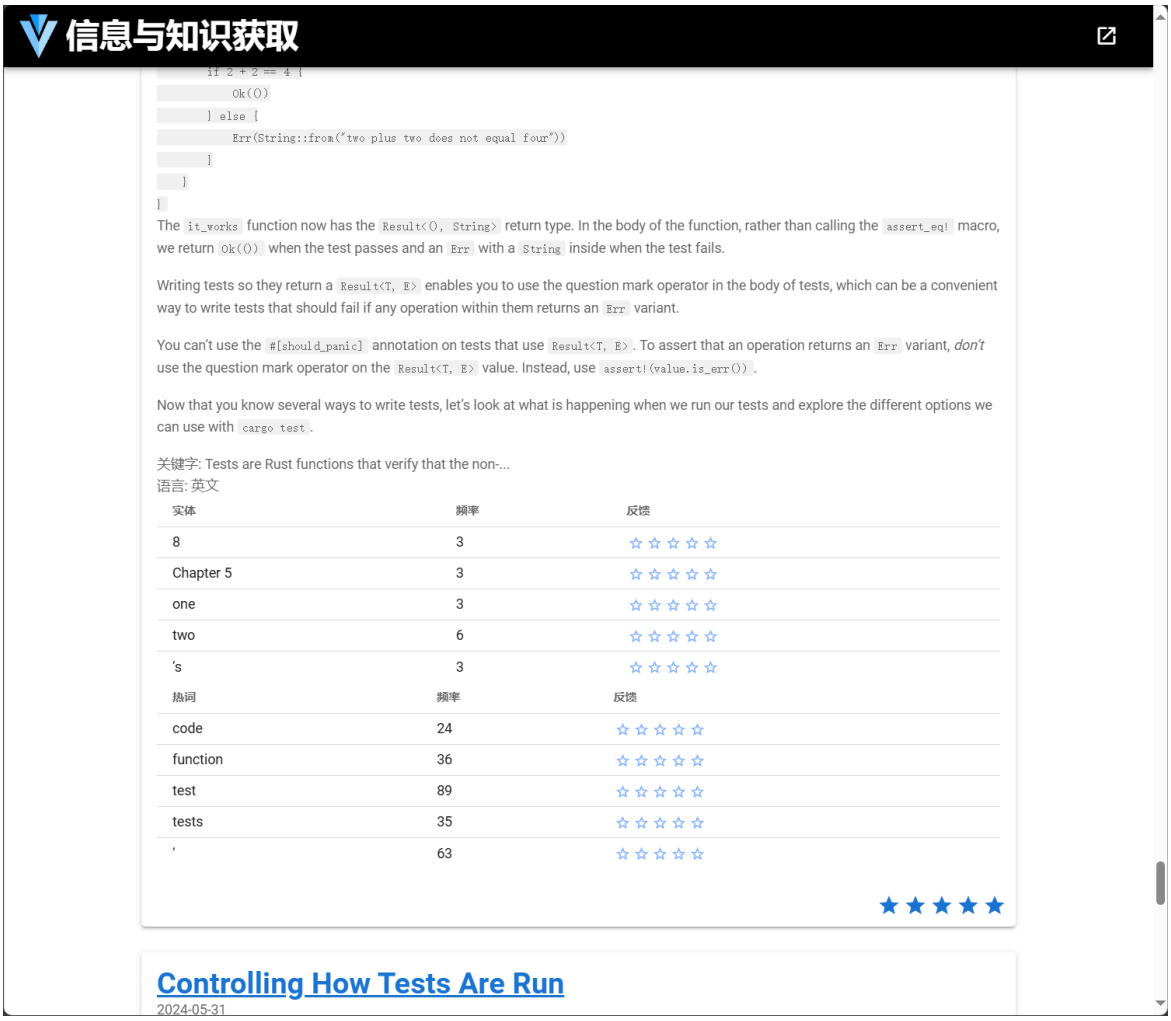


image-20240609160902466

- **组件**：`<v-card>`
- **功能**：
  - 动态展示搜索结果，每个结果为一个卡片，展示包括**文档标题、相关度、介绍信息、链接**（可点击跳转），**右下角**有对每个搜索结果的**评分反馈**，用户可以进行评分。
  - 用户可点击每张卡片的标题，**点击标题将根据该结果的文章 id，从后端调取本结果的详细信息。**（再次点击即可隐藏）
  - 详细信息包括：**整篇文章的所有内容、文章关键字、文章实体表格、文章热词表格。**



- image-20240609162043038



- image-20240609162126049

- **布局**：结果以列表形式排列，通过 Vuetify 的响应式布局确保在不同设备上的显示效果。
- **接口函数**：获取文档详细信息和实体热词表格。

```
toggleDetail(id) {
  // 检查detailMap中是否存在该id且其visible属性为true
  if (this.detailMap[id] && this.detailMap[id].visible) {
    // 如果已经可见，则设置为不可见
    this.$set(this.detailMap[id], "visible", false);
  }
  // 检查detailMap中是否存在该id且其visible属性为false
  else if (this.detailMap[id] && !this.detailMap[id].visible) {
    // 如果不可见，则设置为可见
    this.$set(this.detailMap[id], "visible", true);
  } else {
    // 如果detailMap中没有该id的信息，通过axios请求获取数据
    axios
      .all([
        axios.get(`api/v1/document`, { params: { id } }),  // 请求文档详情
        axios.get(`api/v1/extract_info`, { params: { id } }),  // 请求提取信息
      ])
      .then(
        axios.spread((DocRes, infoRes) => {
          // 处理响应数据
          const entitiesWithScore = Object.entries(infoRes.data.entities).reduce((acc, [key
            // 初始化每个实体的评分为0
            acc[key] = { value, score: 0 };
            return acc;
          }, {});

          const hotWordsWithScore = Object.entries(infoRes.data.hot_words).reduce((acc, [ke
            // 初始化每个热词的评分为0
            acc[key] = { value, score: 0 };
            return acc;
          }, {});

          // 设置detailMap以包含获取的详情数据
          this.$set(this.detailMap, id, {
            visible: true,  // 设置为可见
            content: DocRes.data.content,  // 文档内容
            keywords: DocRes.data.keywords,  // 关键词
            Lang: DocRes.data.Lang,  // 语言
            entities: infoRes.data.entities,  // 实体
            hot_words: infoRes.data.hot_words,  // 热词
          });
        })
      )
```

```
        .catch((error) => {
          // 处理请求错误
          console.error("Error fetching Document details:", error);
        });
    }
  }
```

搜索结果反馈

实体反馈

- **组件**： `<v-rating>`

```
<v-rating
    dense
    hover
    small
    v-model="detailMap[result.Doc.id].entities[key].score"
    @input="handleEntityFeedback(result.Doc.id, key, detailMap[result.Doc.id].entities[key]
></v-rating>
```

实体	频率	反馈
8	3	★★★☆☆
Chapter 5	3	★★☆☆☆
one	3	★★★★★
two	6	☆☆☆☆☆
’s	3	★★★☆☆

image-20240609171828993

- **功能**：允许用户对搜索结果中每个实体的准确性进行评分。
- **属性**：
  - `dense` 和 `small` 使评分组件更紧凑、适合放置在搜索结果卡片中。
  - `hover` 允许用户在鼠标悬停时预览评分效果。
  - `v-model` 绑定到 `detailMap[result.Doc.id].entities[key].score` ，实现数据的双向绑定。
  - `@input` 事件处理函数 `handleEntityFeedback` 发送用户的评分到后端。
- **布局**：每个搜索结果的详细信息区域均设有实体反馈评分组件，与实体信息并排展示。
- **接口函数：**

```
handleEntityFeedback(resultId, item, score) {
  const payload = {
    item,
    resultId,
    score
  };
  axios.post(`api/v1/entity_feedback`, payload)
    .then(response => {
      console.log("Entity Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending entity feedback", error);
    });
},
```

热词反馈

- **组件**：`<v-rating>`

```
<v-rating
    dense
    hover
    small
    v-model="detailMap[result.Doc.id].hot_words[key].score"
    @input="handleHotwordFeedback(result.Doc.id, key, detailMap[result.Doc.id].hot_words[ke
></v-rating>
```

热词	频率	反馈
code	24	☆☆☆☆☆
function	36	★★★☆☆
test	89	★★★★☆
tests	35	★★★☆☆
’	63	★☆☆☆☆

image-20240609171801354

- **功能**：允许用户对搜索结果中的热词进行评分。
- **属性**：
  - `dense` 和 `small` 使评分组件更紧凑、适合放置在搜索结果卡片中。
  - `hover` 允许用户在鼠标悬停时预览评分效果。

    - `v-model` 绑定到 `detailMap[result.Doc.id].hot_words[key].score`，实现数据的双向绑定。
    - `@input` 事件处理函数 `handleEntityFeedback` 发送用户的评分到后端。
- **布局**：与实体反馈类似，热词反馈组件与对应的热词信息并排展示。
- **接口函数**：

```
handleHotwordFeedback(resultId, item, score) {
  const payload = {
    item,
    resultId,
    score
  };
  axios.post(`api/v1/hotword_feedback`, payload)
    .then(response => {
      console.log("Hotword Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending hotword feedback", error);
    });
},
```

整体准确率评价反馈

- **组件**： `<v-rating>`

```
<v-rating
    dense
    hover
    v-model="result.Score"
    @input="handleOverallFeedback(result.Doc.id, result.Score)"
></v-rating>
```

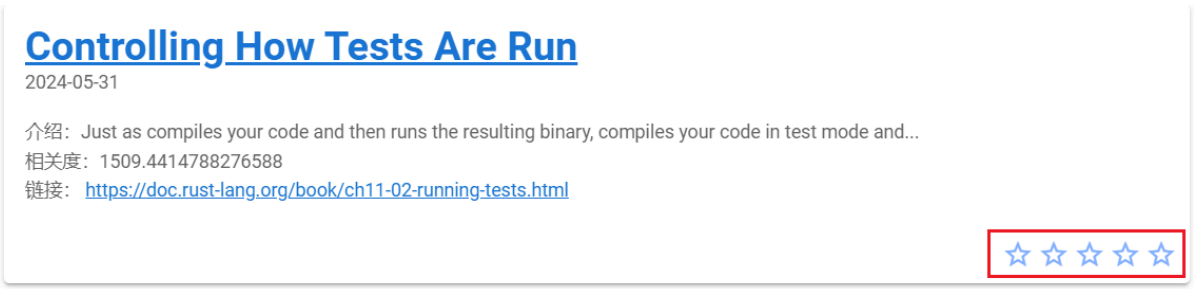


image-20240609171734666

- **功能**：提供对整个搜索结果的整体准确率满意度评价。
- **属性**：
    - `hover` 和 `dense` 属性同上。

    - `v-model` 绑定到 `result.Score`。
    - `@input` 通过 `handleOverallFeedback` 方法发送整体评分数据到后端。
- **布局**：整体评价组件位于搜索结果卡片的底部，方便用户在查看完信息后给出整体准确率评价。
- **接口函数**：

```
handleOverallFeedback(resultId, Score) {
  const payload = {
    resultId,
    Score
  };
  axios.post(`api/v1/feedback`, payload)
    .then(response => {
      console.log("Overall Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending overall feedback", error);
    });
}
```

信息检索服务

信息检索服务是本次实验的核心内容，在实验中，我在后端服务初始化的过程中加载文档并对其进行预处理。在读入文档之后我对文档进行分词，然后构建倒排索引，同时计算并存储 TF-IDF 值，避免查询时重复计算。在查询时为了提高查询的准确率和性能，我通过计算查询向量和文档向量的余弦相似度作为查询结果的关联度，然后根据关联度对查询结果进行排序并返回结果。

加载文档

首先从指定路径中加载文档，文档格式是之前存的 `json` 文件，同时设置每篇文档的基本信息，如 id、语言等。

```
func LoadDocuments(dir string) ([]Document, error) {
    files, err := ioutil.ReadDir(dir)

    var documents []Document

    for _, file := range files {
        if filepath.Ext(file.Name()) != ".json" {
            continue
        }
        filename := filepath.Join(dir, file.Name())

        f, err := os.Open(filename)
        defer f.Close()

        var docs []Document
        dec := json.NewDecoder(f)
        if err := dec.Decode(&docs); err != nil {
            return nil, err
        }

        setLang(docs, filename)

        documents = append(documents, docs...)

        f.Close()
    }

    for id, doc := range documents {
        doc.Id = strconv.Itoa(id)
        sTitle := strconv.QuoteToASCII(doc.Title)
        doc.Title = sTitle[1 : len(sTitle)-1]
        sUnicode := strconv.QuoteToASCII(doc.Content)
        doc.Content = sUnicode[1 : len(sUnicode)-1]
    }

    return documents, nil
}
```

分词

分词是信息检索服务的基础，对于中英文文档，我均使用 `jieba` 库进行分词并去除停用词防止干扰。

```go
func WordSplit(query string) []string {
    defer func() {
        if panicInfo := recover(); panicInfo != nil {
            log.Errorf("%v, %s", panicInfo, string(debug.Stack()))
        }
    }()

    words := seg.Cut(query, true)

    words = filter(words, stopWords)
    for i := range words {
        words[i] = strings.TrimSpace(words[i])

    }

    return words
}

func filter(slice []string, unwanted []string) []string {
    unwantedSet := make(map[string]any, len(unwanted))
    for _, s := range unwanted {
        unwantedSet[s] = struct{}{}
    }

    var result []string
    for _, s := range slice {
        if _, ok := unwantedSet[s]; !ok {
            result = append(result, s)
        }
    }

    return result
}
```

构建索引

索引是查询的基础，在实验中，我对每篇文档均构建了正向索引和倒排索引，记录文章的 id 映射和该文章中出现的所有词语及其位置信息。具体构建流程如下：

1. 遍历所有文章，构建文章 id 对文章内容的映射关系（正向索引）；
2. 对文章进行分词并去除停用词；
3. 遍历文章中的每个关键词，建立一个关键词到文章内容的索引映射（倒排索引）。

```
    for _, doc := range documents {
        idDocMap[doc.Id] = doc

        words := WordSplit(doc.Keywords)
        for _, word := range words {
            if !isStopWord(word) {
                word = strings.ToLower(word)
                docIndex[word] = append(docIndex[word], doc)
            }
        }
    }
```

计算并存储 TF-IDF 值

本次实验使用 TF-IDF 进行特征提取。词频（term frequency, TF）， 指的是某一个给定的词语在该文件中出现的频率。

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

上式中 $n_{i,j}$ 表示第 j 篇文章中出现第 i 个词的频数，而 $\sum_k n_{k,j}$ 表示统计第 j 篇文章中所有词的总数。

逆向文件频率（inverse document frequency, IDF），某一特定词语的 IDF，可以由一类文件中的总文件数目除以该类中包含该词语之文件的数目， 再将得到的商取对数得到。 IDF 是一个词语普遍重要性的度量。

$$IDF_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

上式中 D 表示一个文档的集合，有 $\{d_1, d_2, d_3, \cdots\} \in D$，取模即是计算这个集合中文档的个数。$t_i$ 表示第 i 个词，$j : t_i \in d_i$ 表示第 i 个单词属于文本 d_j，对其取模即是计算包含单词 i 的文本的个数。

TF-IDF 值即是 TF 值与 IDF 值之积， TF-IDF 综合表征了该词在文档中的重要程度和文档区分度。

根据上述算法，我的代码实现如下：

```
// 计算idf
for word := range docIndex {
    if _, ok := idfMap[word]; !ok {
        word = strings.ToLower(word)
        idfMap[word] = math.Log(totalDocs / float64(len(docIndex[word])))
    }
}

// 根据tf-idf计算文档向量，创建tf-idf索引
for _, doc := range documents {
    docVector := buildDocumentVector(doc)
    words := WordSplit(doc.Keywords)

    for _, word := range words {
        word = strings.ToLower(word)
        index[word] = append(index[word], docVector)
    }
}
```

计算文档向量的 `buildDocumentVector` 函数如下：

```
func buildDocumentVector(doc model.Document) DocumentVector {
    vector := make(map[string]float64)
    words := WordSplit(doc.Keywords)
    wordCount := float64(len(words))

    // 计算tf
    for _, word := range words {
        word = strings.ToLower(word)
        vector[word] += 1.0 / wordCount
    }

    // 计算tf-idf
    magnitude := 0.0
    for word, tf := range vector {
        word = strings.ToLower(word)
        tfIdf := tf * idfMap[word]
        vector[word] = tfIdf
        magnitude += tfIdf * tfIdf
    }

    // 归一化
    if magnitude > 0.0 {
        sqrtMagnitude := math.Sqrt(magnitude + epsilon)
        for word := range vector {
            word = strings.ToLower(word)
            vector[word] /= sqrtMagnitude
        }
    }

    return DocumentVector{Doc: doc, Vector: vector}
}
```

检索

本次实验使用余弦相似度来计算两个查询语句和文章的相似度，余弦相似度定义如下：

$$\cos(\theta) = \frac{A \cdot B}{|A|\,|B|}$$

其中 A 和 B 分别是两个向量，θ 是两个向量夹角的弧度。

在项目中我的实现流程如下：

1. 计算查询语句的向量，根据查询词构建 TF-IDF 向量。此过程包括计算每个查询词的 IDF 值，以及归一化查询向量以确保后续相似度计算的准确性；
2. 对于每个查询词，遍历索引中的文档向量，使用 `cosineSimilarity` 函数计算查询向量与文档向量之间的余弦相似度;
3. 根据文档中查询词出现的位置、频率和是否出现在标题中对相似度进行调整，以提高搜索的相关性;
4. 使用协程并行处理这些计算，并通过 channel 收集每个文档的得分，以提高效率；
5. 收集所有得分后，进一步根据查询词在文档中的总出现次数和标题中出现的次数调整最终得分。

详细代码如下：

```
func buildSummaryDocument(doc model.Document) model.SummaryDocument {
    summaryDoc := model.SummaryDocument{
        Id:      doc.Id,
        Title:   doc.Title,
        URL:     doc.URL,
        Date:    doc.Date,
        Content: calculateSummary(doc.Keywords),
    }
    return summaryDoc
}

func SearchIndex(queryWords []string, page, resultsPerPage int) ([]model.SearchResult, error) {
    if len(queryWords) == 0 {
        return nil, errors.New("empty query")
    }

    queryVector := buildQueryVector(queryWords)
    log.Info("queryVector:", queryVector)

    magnigude := 0.0
    for _, tfidf := range queryVector {
        magnigude += tfidf * tfidf
    }
    if magnigude == 0 {
        log.Info("Query made up of words in every or no documents. Returning all documents.")
        results := make([]model.SearchResult, 0, len(docs))
        for _, doc := range docs {
            results = append(results, model.SearchResult{Doc: buildSummaryDocument(doc), Score:
        }

        return results, nil
    }

    vectorCounts := make(map[string]int)
    for _, word := range queryWords {
        word = strings.ToLower(word)
        if vectors, ok := index[word]; ok {
            for _, vector := range vectors {
                vectorCounts[vector.Doc.Id]++
            }
        }
    }
```

```
queryWordCounts := make(map[string]int)
titleQueryWordCounts := make(map[string]int)

var mutex sync.Mutex

scoresChansMap := make(map[string]chan float64)
for id, count := range vectorCounts {
    scoresChansMap[id] = make(chan float64, count)
}

var wg sync.WaitGroup

for _, word := range queryWords {
    word = strings.ToLower(word)
    if vectors, ok := index[word]; ok {
        for _, vector := range vectors {

            wg.Add(1)
            go func(w string, v DocumentVector, scoresChan chan float64) {
                defer wg.Done()

                wi := strings.ToLower(w)

                score := cosineSimilarity(queryVector, v.Vector)

                frequency := float64(len(WordSplit(v.Doc.Keywords)))
                position := float64(strings.Index(v.Doc.Keywords, wi))
                length := float64(len(v.Doc.Keywords))

                adjustment := (1 + math.Log(frequency+1)) * (1 / (1 + math.Log(length+1)) *
                score *= adjustment

                if strings.Contains(v.Doc.Keywords, wi) || strings.Contains(strings.ToLower(
                    mutex.Lock()
                    if strings.Contains(v.Doc.Keywords, wi) {
                        queryWordCounts[v.Doc.Id]++
                    }
                    if strings.Contains(strings.ToLower(v.Doc.Title), wi) {
                        titleQueryWordCounts[v.Doc.Id]++
                    }
                    mutex.Unlock()
                }
                scoresChan <- score
```

```
            }(word, vector, scoresChansMap[vector.Doc.Id])
        }
    }
}

go func() {
    wg.Wait()
    for _, scoresChan := range scoresChansMap {
        close(scoresChan)
    }
}()

scoreMap := make(map[string]*model.SearchResult)
for id, scoresChan := range scoresChansMap {
    totalScore := 0.0
    for score := range scoresChan {
        totalScore += score
    }
    totalScore *= float64(1 + queryWordCounts[id])

    totalScore *= 1.2 * float64(1+titleQueryWordCounts[id])

    summaryDoc := buildSummaryDocument(idDocMap[id])
    scoreMap[id] = &model.SearchResult{Doc: summaryDoc, Score: totalScore}
}

log.Info(len(scoreMap), " results")
log.Debug(">>> scoreMap")
for k, v := range scoreMap {
    log.Debug(k, ":", "Doc:", v.Doc, "Score:", v.Score)
}
log.Debug("<<< scoreMap")

results := make([]model.SearchResult, 0, len(scoreMap))
for _, result := range scoreMap {
    results = append(results, *result)
}

sort.Slice(results, func(i, j int) bool {
    return results[i].Score > results[j].Score
})

start := (page - 1) * resultsPerPage
```

```
    end := start + resultsPerPage
    if start > len(results) {
        start = len(results)
    }
    if end > len(results) {
        end = len(results)
    }

    results = results[start:end]

    return results, nil
}

func buildQueryVector(queryWords []string) map[string]float64 {
    vector := make(map[string]float64)
    wordCount := float64(len(queryWords))

    for _, word := range queryWords {
        word = strings.ToLower(word)
        vector[word] += 1.0 / wordCount
    }

    magnitude := 0.0
    for word, tf := range vector {
        word = strings.ToLower(word)
        idf, ok := idfMap[word]
        if !ok {
            continue
        }
        tfIdf := idf * tf
        vector[word] = tfIdf
        magnitude += tfIdf * tfIdf
    }

    if magnitude > 0.0 {
        sqrtMagnitude := math.Sqrt(magnitude + epsilon)
        for word := range vector {
            vector[word] /= sqrtMagnitude
        }
    }

    return vector
}
```

```
func cosineSimilarity(vector1, vector2 map[string]float64) float64 {
    dotProduct := 0.0
    magnitude1 := 0.0
    magnitude2 := 0.0
    for word, value := range vector1 {
        word = strings.ToLower(word)
        dotProduct += value * vector2[word]
        magnitude1 += value * value
    }
    for _, value := range vector2 {
        magnitude2 += value * value
    }

    sqrtEpsMag1 := math.Sqrt(magnitude1 + epsilon)
    sqrtEpsMag2 := math.Sqrt(magnitude2 + epsilon)
    return dotProduct / (sqrtEpsMag1 * sqrtEpsMag2)
}

func calculateSummary(content string) string {
    if len(content) > 100 {
        return content[:100] + "..."
    }
    return content
}
```

多媒体关键词提取服务

想要实现多媒体的检索服务，首先就要解决从多媒体中提取出关键词的问题，由于 python 在这一方面具有优势，因此我们选择使用 python 完成这一任务。

具体流程是：

1. python 使用 flask 框架接受请求；
2. 将图片输入到 ResNet50 模型中进行对象识别；
3. 将前五个最可能的对象转换成关键词返回；

接口代码如下：

```python
@app.route("/image_to_keywords", methods=["POST"])
def image_to_keywords():
    if "file" not in request.files:
        return "No file part", 400
    file = request.files["file"]

    if file.filename == "":
        return "No selected file", 400
    log.info(file.filename)
    result, code = image_detection.image_to_keywords(file)
    if code != 200:
        log.error(result)
        return result, code
    return result, code
```

模型处理代码如下（错误处理代码则删去不再展示）：

```
config = tf.compat.v1.ConfigProto(
    gpu_options=tf.compat.v1.GPUOptions(allow_growth=True))
sess = tf.compat.v1.Session(config=config)

log = logging.getLogger("ImageToKeywords")

# Image object detection model
model = ResNet50(weights="imagenet")


def image_to_keywords(file: str) -> tuple[str, int]:
    if file.filename == "":
        return ("No selected file", 400)
    log.info(file.filename)

    img = (
        Image.open(io.BytesIO(file.read()))
        .convert("RGB")
        .resize((224, 224))
    )

    x = img_to_array(img)

    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    preds = model.predict(x)
    predictions = decode_predictions(preds, top=5)[0]

    if len(predictions) >= 3:
        keywords = [pred[1] for pred in predictions[:3]]
    else:
        keywords = [pred[1] for pred in predictions]
    keywords = " ".join(kw for kw in keywords)
    log.info(keywords)
    return (json.dumps({"keyword": keywords}), 200)
```

检索结果评价

实体反馈

- **组件**：`<v-rating>`

```
<v-rating
    dense
    hover
    small
    v-model="detailMap[result.Doc.id].entities[key].score"
    @input="handleEntityFeedback(result.Doc.id, key, detailMap[result.Doc.id].entities[key]
></v-rating>
```

实体	频率	反馈
8	3	★★★☆☆
Chapter 5	3	★★☆☆☆
one	3	★★★★★
two	6	☆☆☆☆☆
's	3	★★★☆☆

image-20240609171828993

- **功能**：允许用户对搜索结果中每个实体的准确性进行评分。
- **属性**：
  - `dense` 和 `small` 使评分组件更紧凑、适合放置在搜索结果卡片中。
  - `hover` 允许用户在鼠标悬停时预览评分效果。
  - `v-model` 绑定到 `detailMap[result.Doc.id].entities[key].score` ，实现数据的双向绑定。
  - `@input` 事件处理函数 `handleEntityFeedback` 发送用户的评分到后端。
- **布局**：每个搜索结果的详细信息区域均设有实体反馈评分组件，与实体信息并排展示。
- **接口函数：**

```
handleEntityFeedback(resultId, item, score) {
  const payload = {
    item,
    resultId,
    score
  };
  axios.post(`api/v1/entity_feedback`, payload)
    .then(response => {
      console.log("Entity Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending entity feedback", error);
    });
},
```

热词反馈

- **组件**： `<v-rating>`

```
<v-rating
    dense
    hover
    small
    v-model="detailMap[result.Doc.id].hot_words[key].score"
    @input="handleHotwordFeedback(result.Doc.id, key, detailMap[result.Doc.id].hot_words[ke
></v-rating>
```

热词	频率	反馈
code	24	☆☆☆☆☆
function	36	★★★☆☆
test	89	★★★★☆
tests	35	★★★☆☆
’	63	★☆☆☆☆

image-20240609171801354

- **功能**：允许用户对搜索结果中的热词进行评分。
- **属性**：
  - `dense` 和 `small` 使评分组件更紧凑、适合放置在搜索结果卡片中。
  - `hover` 允许用户在鼠标悬停时预览评分效果。
  - `v-model` 绑定到 `detailMap[result.Doc.id].hot_words[key].score` ，实现数据的双向绑定。
  - `@input` 事件处理函数 `handleEntityFeedback` 发送用户的评分到后端。
- **布局**：与实体反馈类似，热词反馈组件与对应的热词信息并排展示。
- **接口函数**：

```
handleHotwordFeedback(resultId, item, score) {
  const payload = {
    item,
    resultId,
    score
  };
  axios.post(`api/v1/hotword_feedback`, payload)
    .then(response => {
      console.log("Hotword Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending hotword feedback", error);
    });
},
```

整体准确率评价反馈

- **组件**：`<v-rating>`

```
<v-rating
    dense
    hover
    v-model="result.Score"
    @input="handleOverallFeedback(result.Doc.id, result.Score)"
></v-rating>
```

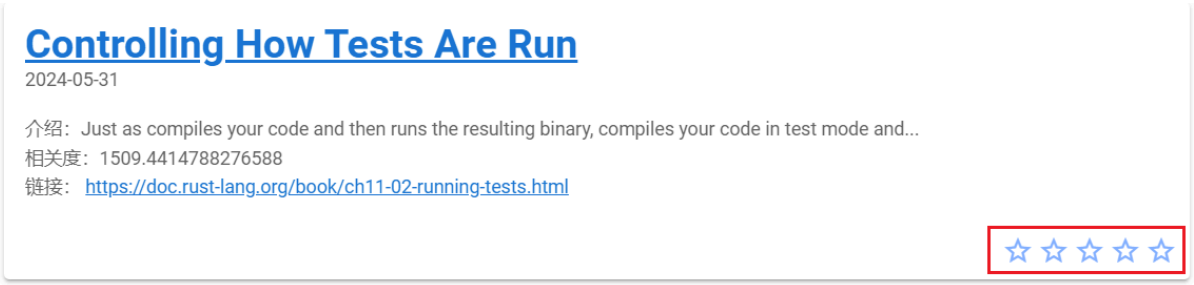


image-20240609171734666

- **功能**：提供对整个搜索结果的整体准确率满意度评价。
- **属性**：
  - `hover` 和 `dense` 属性同上。
  - `v-model` 绑定到 `result.Score` 。
  - `@input` 通过 `handleOverallFeedback` 方法发送整体评分数据到后端。
- **布局**：整体评价组件通常位于搜索结果卡片的底部，方便用户在查看完信息后给出整体准确率评价。

- **接口函数**:

```
handleOverallFeedback(resultId, Score) {
  const payload = {
    resultId,
    Score
  };
  axios.post(`api/v1/feedback`, payload)
    .then(response => {
      console.log("Overall Feedback sent successfully", response);
    })
    .catch(error => {
      console.error("Error sending overall feedback", error);
    });
}
```

优化与创新性

在本次实验中，通过不断地打磨我们的项目，我们实现了以下优化和创新：

后端-算法优化

- 缓存优化：后端自己实现了 LRU 缓存，在内存中缓存每次查询的结果，减少了重复的查询计算，提高了查询的性能和效率；
- TF-IDF 归一化：在计算 TF-IDF 值的时候，我对其进行了归一化，减少了文章长度的影响，在使用时我也加了一个小常数 ε 避免 TF-IDF 值为 0；
- 空间向量查询：在检索比较时，我没有使用简单的线性比较，而是通过计算向量余弦值来计算两个空间向量的相似度，这样提高了检索的性能指标；
- 并发优化：在检索时我通过 Go 的协程并行计算每篇文档的得分，这样充分利用了 Go 轻量级协程的优势，大大提高了检索的性能；
- 基于用户反馈动态修改排名：对于每个查询结果，我们都设置用户可以对其进行反馈，并且通过用户反馈修改查询结果的权重，动态地调整查询结果的排名；
- 多媒体检索：在项目中我们引入了图片识别模型，通过识别提取图片关键词，从而实现多媒体检索。

前端

优化措施

- **异步数据处理**：使用 Vue.js 的异步组件和 `axios` 进行数据请求，优化了页面加载速度和响应时间，避免了在请求数据时阻塞用户界面的问题。
- **组件化开发**：利用 Vue.js 的组件化能力，将前端界面分解为可重用的组件，如搜索栏、结果卡片、反馈评分等。这种方式不仅提高了代码的可维护性，也简化了功能的扩展。
- **用户体验优化**：
  - 在关键组件中实施了响应式设计，确保在不同设备上都能提供良好的用户体验。
  - 通过细致的动画和过渡效果增强了界面的交互性，如加载动画和按钮点击反馈。
  - 针对用户操作提供即时的反馈信息，比如在发送反馈评分后，通过控制台日志确认反馈已成功发送。
- **错误处理与数据验证**：
  - 在数据输入和网络请求中实施了全面的错误处理机制，确保了应用的稳定运行和用户的流畅体验。
  - 对用户输入进行验证，防止无效或恶意的数据被提交到后端。

创新性特点

- **集成的反馈机制**：引入了多层次的用户反馈系统，包括实体评分、热词评分和整体搜索结果评分。这不仅使用户能够直接参与改善搜索结果，也为算法优化提供了实时数据。
- **多模态搜索功能**：实现了关键词搜索与图像搜索的结合，提供了一种多模态的信息检索方式。用户可以通过文本或图像中的内容进行搜索，增强了搜索的灵活性和准确性。
- **实时动态更新**：通过 Vue.js 的双向数据绑定和组件状态管理，实现了搜索结果和用户反馈的实时动态更新。用户在界面上的任何操作都可以即时反映，提高了交互的实时性。

环境和社会可持续发展思考

环境影响

- **减少能源消耗**：信息检索系统通过自动化处理大量数据，可能会对服务器造成较大负荷，进而影响能源消耗。为了降低这种影响，我们采用了能效较高的数据处理算法和节能的服务器配置。此外，通过优化爬虫的爬取策略，减少不必要的数据请求，可以进一步减轻对源网站服务器的压力，间接减少整个网络中的能源消耗。
- **绿色技术选择**：在选择第三方服务和云平台时，优先考虑那些承诺使用可再生能源并具有良好碳足迹记录的供应商。例如，使用支持绿色能源的数据中心可以减少项目的环境负担。

社会效益

- **信息获取的公平性**：本信息检索系统支持中英文内容的处理，有助于跨语言和文化的信息流通，促进了知识的平等获取。这对于教育资源的公平分配尤其重要，可以帮助来自不同背景的用户访问和利用全球的信息资源。

- **促进知识共享和教育**：系统提供的多媒体关键词提取服务和自然语言查询功能，使用户能够更容易地找到所需信息，从而促进知识的传播和教育的普及。这种技术的应用尤其可以支持教育不发达地区的学习和研究，减少城乡之间的教育差异。
- **提高社会意识和参与**：通过提供高效的信息检索工具，可以增强公众对于重要社会问题的意识和理解，比如环境保护、公共健康和社会正义等。用户可以更容易地获取相关信息，从而在这些重要问题上做出更为明智的决策和参与。

技术与可持续性的结合

- 在技术实现方面，本项目特别重视环保和可持续性原则。通过采用最新的算法优化技术，我们显著提高了数据处理效率，从而减少了能耗。同时，项目在选择服务器和存储解决方案时，优先考虑那些采用可再生能源的服务提供商。这种策略不仅降低了系统的环境足迹，也体现了我们对环境保护的承诺。

促进包容性和平等

- 信息检索系统设计之初就考虑到了多样性和包容性，特别是在语言处理功能上。系统支持中英文的自然语言处理，确保了不同语言用户的信息获取需求得到满足。此外，我们通过用户界面的多语言支持和无障碍设计，使系统对不同文化和能力水平的用户都友好，从而推动了信息获取的平等性和包容性。

实验总结

本次实验的核心目标是设计并实现一个多功能的信息检索系统，该系统能够处理中英文文档的检索，并提供准确的搜索结果。实验过程中，我们面临了各种挑战，同时也获得了宝贵的知识与经验。

1. **数据爬取与处理：** 我们使用了 Scrapy 框架来爬取网络数据，并存储为 json 格式。处理数据时，需要对中文进行分词，这一步骤对准确建立倒排索引至关重要。我们选择使用结巴分词框架，它在处理中文分词时表现出色，但同时也需要调整参数以适应不同的语境与专业术语。
2. **倒排索引与搜索算法的实现：** 建立高效的倒排索引并实现基于 TF-IDF 的向量空间模型匹配算法是另一个挑战。这要求我们不仅要在理论上掌握信息检索的基本原理，还要在实践中调整和优化算法。
3. **多媒体检索服务：** 利用 Tensorflow 和 Flask 框架开发图片检索功能，我们需要处理大量的图像数据，并从中提取关键词进行检索。这一过程中的图像识别与关键词提取对计算资源和算法优化提出了更高的要求。

通过本项目，我们深入学习了信息检索的核心技术和算法。在实际编码和实现过程中，我们对 Scrapy 和结巴分词的应用有了更深入的理解。同时，通过实际操作 Tensorflow 框架，我们增强了对机器学习在图像处理应用中的理解和技能。此外，项目的多语言处理能力也让我们了解到语言处理的复杂性和技术的多样性。

本信息检索系统成功实现了从数据爬取到用户界面的全流程服务。系统能够有效地对输入的查询进行处理，并按照相关度返回准确的搜索结果。特别是在中文文档的处理上，通过优化分词和索引构建过程，提高了检索的准确率和效率。在项目展示和评审中，该系统得到了用户和专家的肯定，验证了我们设计和实现的有效性。总之，这次实验不仅提升了我们的技术能力，也加深了我们对信息检索领域的理解。

实验分工

	郭晨旭	**韩景锐**
学号	2021211184	2021211176
代码	Go 后端，Scrapy 爬虫，Python 图片提取关键词服务	Vue 前端
报告	项目概述，后端设计与实现，信息检索服务， 多媒体关键词提取服务，优化与创新	前端设计与实现，优化与创新， 环境和社会可持续发展思考，实验总结