聚合搜索平台项目笔记

项目介绍

一个企业级聚合搜索平台（简化版的搜索中台）。

项目的意义：

用户角度：允许用户在同一个页面集中搜索出不同来源、不同类型的内容，提升用户的检索效率和搜索体验。

企业角度：当企业有多个项目的数据需要被搜索时，无需针对每个项目单独开发搜索功能，可以直接将数据接入搜索中台，提升开发效率。

特点

1. 比较新颖，且项目周期不会很长（计划一个月），争取带大家快速搞定，成为简历上的亮眼项目
2. 选用 Vue 3 + Spring Boot 2.7 新版本实现，包含完整的前后端，并且依然是从 0 到 1 带大家开发
3. 前端讲得非常细；后端也包含较多的编程技巧，会带大家学习实践数据抓取 => 数据同步 => 搜索引擎等一系列业务开发
4. 项目中会分享很多架构设计知识，带大家提升一个思考的 level

项目资料

完整源码见星球代码库：https://t.zsxq.com/08DElcxT7 <https://t.zsxq.com/08DElcxT7> （前端：yuso-frontend，后端：yuso-backend）

全部直播回放 + 大纲：https://www.code-nav.cn/course/1790979621621641217/section/1790981240778174466?type= <https://www.code-nav.cn/course/1790979621621641217/section/1790981240778174466?type=>

第一期

单集直播回放：https://t.zsxq.com/0cVj0mO5v <https://t.zsxq.com/0cVj0mO5v>

主要内容：

1. 项目介绍：背景、技术栈、业务流程、项目计划
2. 前端项目初始化
3. 后端项目初始化
4. 万用项目模板介绍
5. 前端聚合搜索页面开发
6. 前后端联调

技术栈介绍

全栈项目

前端

- Vue
- Ant Design Vue
- Lodash

后端

- Spring Boot
- MySQL
- Elasticsearch （Elastic Stack）搜索引擎
- 数据抓取
  - 离线
  - 实时
- 数据同步（4 种同步方式）
  - 定时
  - 双写

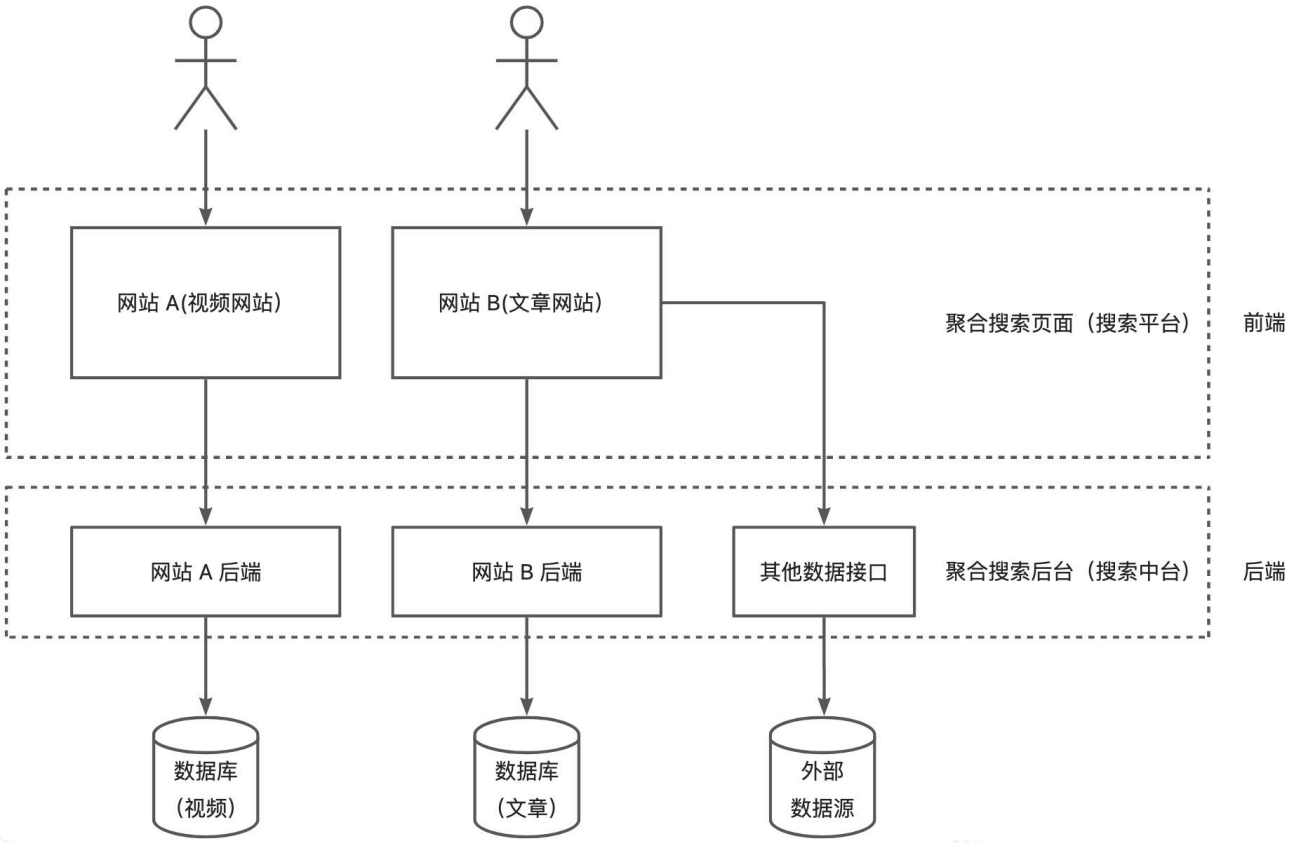
    - Logstash
    - Canal
- JMeter 压力测试
- Guava Retrying 保证 API 的稳定性?

业务流程

1. 先得到各种不同分类的数据
2. 提供一个搜索页面（单一搜索 + 聚合搜索），支持搜索

还可以做一些优化，比如关键词高亮、搜索建议、防抖节流：

项目架构图：



前端初始化

步骤：

1. 参考 Ant Design 组件库的官方文档来搭建初始化项目（https://2x.antdv.com/docs/vue/getting-started-cn <https://2x.antdv.com/docs/vue/getting-started-cn> ），并整合组件库。

2．删减不需要的页面和路由

后端初始化

步骤：

1．使用星球 springboot-init 万用项目模板

2．直接使用 swagger 文档在线测试接口

前端开发

组件库使用方式：从上到下依次在组件库文档中找到对应组件，复制粘贴 + 修改，完成基本页面开发。

记录搜索状态

目标：用 url 记录页面搜索状态，当用户刷新页面时，能够从 url 还原之前的搜索状态

需要双向同步：url <=> 页面状态

核心小技巧：把同步状态改成单向，即只允许 url 来改变页面状态，不允许反向

分步骤来实现，思路更清晰：

1．让用户在操作的时候，改变 url 地址（点击搜索框，搜索内容填充到 url 上？切换 tab 时，也要填充）

2．当 url 改变的时候，去改变页面状态（监听 url 的改变）

前后端联调

使用 Axios 向后端发送请求。

步骤（请参考官方文档：https://www.axios-http.cn/docs/intro <https://www.axios-http.cn/docs/intro> ）：

1．前端整合 Axios

2．自定义 Axios 实例

3．发送请求

第二期

单集直播回放：https://t.zsxq.com/0ck9qfV1v <https://t.zsxq.com/0ck9qfV1v>

主要内容：

1. 多数据源获取（包含几种爬虫方式的讲解和实践）
    a. 文章（内部）
    b. 用户（内部）
    c. 图片（外部，不是我们自己的项目、自己的用户生产的数据）
2. 前后端接口调通
3. 聚合搜索业务场景分析
4. 聚合搜索接口开发

获取不同类型的数据源

数据抓取流程

1. 分析数据源，怎么获取？
2. 拿到数据后，怎么处理？
3. 写入数据库等存储

数据抓取的几种方式

1. 直接请求数据接口（最方便），可使用 HttpClient、OKHttp、RestTemplate、Hutool（https://hutool.cn/ <https://hutool.cn/> ）等客户端发送请求
2. 等网页渲染出明文内容后，从前端完整页面中解析出需要的内容
3. 有一些网站可能是动态请求的，他不会一次性加载所有的数据，而是要你点某个按钮、输入某个验证码才会显示出数据。可使用无头浏览器：selenium、node.js puppeteer

注意，爬虫技术不能滥用，千万不要给别人的系统造成压力、不要侵犯他人权益！

1、获取文章

内部没有，可以从互联网上获取基础数据 => 爬虫

可使用该网站进行测试抓取：https://www.code-nav.cn/learn/passage <https://www.code-nav.cn/learn/passage> （注意！仅做测试，不要频繁请求！！！），获取到文章后要入库。

离线抓取：定时获取或者只获取一次

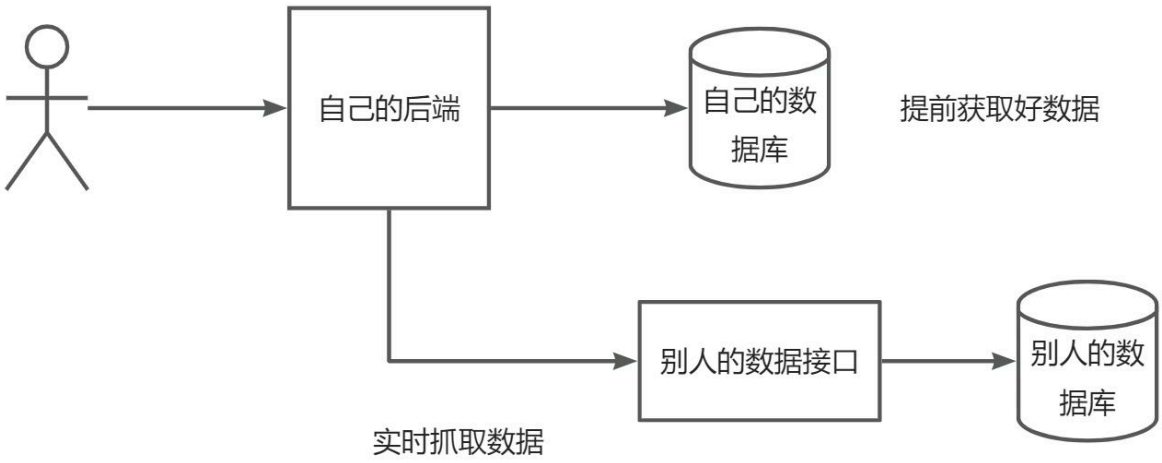
2、用户获取

每个网站的用户基本都是自己的，一般无需从站外获取。

3、图片获取

实时抓取：我们自己的数据库不存这些数据，用户要搜的时候，直接从别人的接口（网站 / 数据库）去搜。

流程如图：



jsoup 解析库：支持发送请求获取到 HTML 文档，然后从中解析出需要的字段。

现有业务场景分析

目前是在页面加载时，调用三个接口分别获取文章、图片、用户数据。

几种不同的业务场景：

1）其实可以用户点某个 tab 的时候，只调用这个 tab 的接口，比如：https://www.code-nav.cn/search/all?current=2&pageSize=8&searchText=&sortField=_score&sortOrder=descend

<https://www.code-nav.cn/search/all?current=2&pageSize=8&searchText=&sortField=_score&sortOrder=descend>)

2）如果是针对聚合内容的网页，其实可以一个请求搞定，比如：https://tophub.today/ <https://tophub.today/>

3）有可能还要查询其他的信息，比如其他数据的总数，同时给用户反馈，比如 B 站搜索页

你要根据实际情况去选择方式！

目前设计存在的问题：

1. 请求数量比较多，可能会收到浏览器的限制
2. 请求不同接口的参数可能不一致，增加前后端沟通成本
3. 前端写调用多个接口的代码，重复代码

聚合接口

1）请求数量比较多，可能会收到浏览器的限制 => 用一个接口请求完所有的数据（后端可以并发，几乎没有并发数量限制）

```
{
  user =  userService.query
  post = postService.query
  picture = pictureService.query
  return user + post + picture
  }
```

2）请求不同接口的参数可能不一致，增加前后端沟通成本 => 用一个接口把请求参数统一，前端每次传固定的参数，后端去对参数进行转换

```
{
   前端统一传 searchText
   后端把 searchText 转换为 userName => queryUser
  }
```

统一返回结果：比如都使用 Page 页面封装

3）前端写调用多个接口的代码，重复代码 => 用一个接口，通过不同的参数去区分查询的数据源

```
{
  前端传 type 调用后端同一个接口，后端根据 type 调用不同的 service 查询
  比如：type =  user，userService.query
  }
```

注意，并发不一定更快！可能存在短板效应。要以实际测试结果为准！

第三期

单集直播回放：https://t.zsxq.com/0cKPUkuOa <https://t.zsxq.com/0cKPUkuOa>

主要内容：

1. 搜索接口优化（运用 3 种设计模式）
2. 前端调整搜索接口调用
3. 从 0 开始学习 Elastic Stack
    a. Elasticsearch 安装
    b. 入门实践
    c. Kibana

聚合接口优化

思考：怎么样能让前端又能一次搜出所有数据、又能够分别获取某一类数据（比如分页场景）

解决方案：

新增 type 字段：前端传 type 调用后端同一个接口，后端根据 type 调用不同的 service 查询

比如前端传递 type = user，后端执行 userService.query

逻辑；

1. 如果 type 为空，那么搜索出所有的数据
2. 如果 type 不为空
    a. 如果 type 合法，那么查出对应数据
    b. 否则报错

问题：type 增多后，要把查询逻辑堆积在 controller 代码里么？

思考：怎么能让搜索系统 **更轻松地** 接入更多的数据源？

门面模式

介绍：帮助我们用户（客户端）去更轻松地实现功能，不需要关心门面背后的细节。

聚合搜索类业务基本都是门面模式：即前端不用关心后端从哪里、怎么去取不同来源、怎么去聚合不同来源的数据，更方便地获取到内容。

当调用你系统（接口）的客户端觉得麻烦的时候，你就应该思考，是不是可以抽象一个门面了。

适配器模式

1）定制统一的数据源接入规范（标准）：

- 什么数据源允许接入?
- 你的数据源接入时要满足什么要求?
- 需要接入方注意什么事情?

本系统要求：任何接入我们系统的数据，它必须要能够根据关键词搜索、并且支持分页搜索。

通过声明接口的方式来定义规范。

2）假如说我们的数据源已经支持了搜索，但是原有的方法参数和我们的规范不一致，怎么办?

使用适配器模式：通过转换，让两个系统能够完成对接。

注册器模式（本质也是单例）

提前通过一个 map 或者其他类型存储好后面需要调用的对象。

效果：替代了 if... else...，代码量大幅度减少，可维护可扩展。

搜索优化

现有问题：搜索不够灵活。

比如搜 “鱼皮rapper” 无法搜到 “鱼皮是 rapper”，因为 MySQL 数据库的 like 是包含查询。

需要分词搜索

Elastic Stack（一套技术栈）

官网：https://www.elastic.co/cn/ <https://www.elastic.co/cn/>

包含了数据的整合 => 提取 => 存储 => 使用，一整套!

各组件介绍：

- beats 套件：从各种不同类型的文件 / 应用中采集数据。比如：a,b,c,d,e,aa,bb,cc
- Logstash：从多个采集器或数据源来抽取 / 转换数据，向 es 输送。比如：a,bb,cc
- elasticsearch：存储、查询数据

- kibana：可视化 es 的数据

安装 ES

elasticsearch：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/setup.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/setup.html>

https://www.elastic.co/guide/en/elasticsearch/reference/7.17/zip-windows.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/zip-windows.html>

只要是一套技术，所有版本必须一致！！！此处都用 7.17 版本。

安装 Kibana

kibana：https://www.elastic.co/guide/en/kibana/7.17/introduction.html <https://www.elastic.co/guide/en/kibana/7.17/introduction.html>

https://www.elastic.co/guide/en/kibana/7.17/install.html <https://www.elastic.co/guide/en/kibana/7.17/install.html>

Elasticsearch 概念

你就把它当成 MySQL 一样的数据库去学习和理解。

入门学习：

- Index 索引 => MySQL 里的表（table）
- 建表、增删改查（查询需要花费的学习时间最多）
- 用客户端去调用 ElasticSearch（3 种）
- 语法：SQL、代码的方法（4 种语法）

ES 相比于 MySQL，能够自动帮我们做分词，能够非常高效、灵活地查询内容。

索引（倒排索引）

正向索引：理解为书籍的目录，可以快速帮你找到对应的内容（怎么根据页码找到文章）

倒排索引：怎么根据内容找到文章

文章 A：你好，我是 rapper

文章 B：鱼皮你好，我是 coder

切词：

你好，我是，rapper

鱼皮，你好，我是，coder

构建倒排索引表：

词	内容 id
你好	文章 A，B
我是	文章 A，B
rapper	文章 A
鱼皮	文章 B
coder	文章 B

用户搜：“鱼皮 rapper”

ES 先切词：鱼皮，rapper

然后去倒排索引表找对应的文章

ES 的几种调用方式

1）restful api 调用（http 请求）

GET 请求：http://localhost:9200/ <http://localhost:9200/>

curl 可以模拟发送请求：curl -X GET "localhost:9200/?pretty"

ES 的启动端口

1. 9200：给外部用户（给客户端调用）的端口
2. 9300：给 ES 集群内部通信的（外部调用不了的）

2）kibana devtools

自由地对 ES 进行操作（本质也是 restful api）

devtools 不建议生产环境使用

3）客户端调用

java 客户端、go 客户端等。

参考文档：https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/_getting_started.html <https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/_getting_started.html>

ES 的语法

DSL

json 格式，好理解；和 http 请求最兼容，应用最广，也是鱼皮个人比较推荐的

建表、插入数据

```
POST post/_doc/ro1R-oYBDwC2db8WGdlL
{
  "title": "鱼皮2",
  "desc": "鱼皮的描述2"
}
```

查询

DSL 语法：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl.html>（忘了就查，不用背）

```
GET post/_search
{
  "query": {
    "match_all": { }
  },
  "sort": [
    {
      "@timestamp": "desc"
    }
  ]
}
```

根据 id 查询：

```
GET post/_doc/ro1R-oYBDwC2db8WGdlL
```

修改

```
POST post/_doc/ro1R-oYBDwC2db8WGdlL
{
  "title": "鱼皮2",
  "desc": "鱼皮的描述2"
}
```

删除

删除普通索引：

```
DELETE index_name
```

删除数据流式索引：

```
DELETE _data_stream/logs-my_app-default
```

EQL

专门查询 ECS 文档（标准指标文档）的数据的语法，更加规范，但只适用于特定场景（比如事件流）

文档：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/eql.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/eql.html>

示例：

```
POST my_event/_doc
{
  "title": "鱼333333皮",
  "@timestamp": "2099-05-06T16:21:15.000Z",
  "event": {
    "original": "192.0.2.42 - - [06/May/2099:16:21:15 +0000] \"GET /images/bg
  }
}


GET my_event/_eql/search
{
  "query": """
    any where 1 == 1
  """
}
```

SQL

文档：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/sql-getting-started.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/sql-getting-started.html>

学习成本低，但是可能需要插件支持、 **性能较差**

示例：

```
POST /_sql?format=txt
{
  "query": "SELECT * FROM post where title like '%鱼皮%'"
}
```

Painless Scripting language

编程式取值，更灵活，但是学习成本高

Mapping

文档：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/explicit-mapping.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/explicit-mapping.html>

可以理解为数据库的表结构，有哪些字段、字段类型。

ES 支持动态 mapping，表结构可以动态改变，而不像 MySQL 一样必须手动建表，没有的字段就不能插入。

显示创建 mapping：

```
GET user/_mapping

PUT user
{
  "mappings": {
    "properties": {
      "age":    { "type": "integer" },
      "email":  { "type": "keyword"  },
      "name":   { "type": "text"  }
    }
  }
}
```

第四期

单集直播回放：https://t.zsxq.com/0crCWmhHC <https://t.zsxq.com/0crCWmhHC>

主要内容：

1. ES 搜索引擎实战（2 种 Java 客户端操作方式）
2. 数据同步实战（4 种同步方式）
3. Kibana 搭建看板
4. JMeter 接口性能测试
5. 其他扩展思路

ElasticStack 概念

ES 索引（Index）=> 表

ES field（字段）=> 列

倒排索引

调用语法（DSL、EQL、SQL 等）

Mapping 表结构

- 自动生成 mapping
- 手动指定 mapping

分词器

指定了分词的规则。

内置分词器：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/analysis-analyzers.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/analysis-analyzers.html>

空格分词器：whitespace，结果 The、quick、brown、fox.

示例：

```
POST _analyze
{
  "analyzer": "whitespace",
  "text":     "The quick brown fox."
}
```

标准分词规则，结果：is、this、deja、vu

```
POST _analyze
{
  "tokenizer": "standard",
  "filter":  [ "lowercase", "asciifolding" ],
  "text":      "Is this déja vu?"
}
```

关键词分词器：就是不分词，整句话当作专业术语

```
POST _analyze
{
  "analyzer": "keyword",
  "text":     "The quick brown fox."
}
```

IK 分词器（ES 插件）

中文友好：https://github.com/medcl/elasticsearch-analysis-ik <https://github.com/medcl/elasticsearch-analysis-ik>

下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases/tag/v7.17.7 <https://github.com/medcl/elasticsearch-analysis-ik/releases/tag/v7.17.7> （注意版本一致）

思考：怎么样让 ik 按自己的想法分词?

解决方案：自定义词典（自己尝试）

ik_smart 和 ik_max_word 的区别？举例：“小黑子”

ik_smart 是智能分词，尽量选择最像一个词的拆分方式，比如“小”、“黑子”

ik_max_word 尽可能地分词，可以包括组合词，比如 “小黑”、“黑子”

打分机制

比如有 3 条内容：

1. 鱼皮是狗
2. 鱼皮是小黑子
3. 我是小黑子

用户搜索：

1. 鱼皮，第一条分数最高，因为第一条匹配了关键词，而且更短（匹配比例更大）
2. 鱼皮小黑子 => 鱼皮、小、黑子，排序结果：2 > 3 > 1

参考文章：https://liyupi.blog.csdn.net/article/details/119176943 <https://liyupi.blog.csdn.net/article/details/119176943>

官方参考文章：https://www.elastic.co/guide/en/elasticsearch/guide/master/controlling-relevance.html <https://www.elastic.co/guide/en/elasticsearch/guide/master/controlling-relevance.html>

ES 调用方式

3 种：

1. HTTP Restful 调用
2. kibana 操作（dev tools）
3. 客户端操作（Java）

Java 操作 ES

3 种方式：

1）ES 官方的 Java API

https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/introduction.html <https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/introduction.html>

快速开始：https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/connecting.html <https://www.elastic.co/guide/en/elasticsearch/client/java-api-client/7.17/connecting.html>

2）ES 以前的官方 Java API，HighLevelRestClient（已废弃，不建议用）

3）Spring Data Elasticsearch（推荐）

spring-data 系列：spring 提供的操作数据的框架

spring-data-redis：操作 redis 的一套方法

spring-data-mongodb：操作 mongodb 的一套方法

spring-data-elasticsearch：操作 elasticsearch 的一套方法

官方文档：https://docs.spring.io/spring-data/elasticsearch/docs/4.4.10/reference/html/ <https://docs.spring.io/spring-data/elasticsearch/docs/4.4.10/reference/html/>

自定义方法：用户可以指定接口的方法名称，框架帮你自动生成查询

用 ES 实现搜索接口

1、建表（建立索引）

数据库表结构：

```sql
-- 帖子表
create table if not exists post
(
    id         bigint auto_increment comment 'id' primary key,
    title      varchar(512)                       null comment '标题',
    content    text                               null comment '内容',
    tags       varchar(1024)                      null comment '标签列表（json
    thumbNum   int      default 0                 not null comment '点赞数',
    favourNum  int      default 0                 not null comment '收藏数',
    userId     bigint                             not null comment '创建用户 i
    createTime datetime default CURRENT_TIMESTAMP not null comment '创建时间',
    updateTime datetime default CURRENT_TIMESTAMP not null on update CURRENT_
    isDelete   tinyint  default 0                 not null comment '是否删除',
    index idx_userId (userId)
) comment '帖子' collate = utf8mb4_unicode_ci;
```

ES Mapping：

id（可以不放到字段设置里）

ES 中，尽量存放需要用户筛选（搜索）的数据

aliases：别名（为了后续方便数据迁移）

字段类型是 text，这个字段是可被分词的、可模糊查询的；而如果是 keyword，只能完全匹配、精确查询。

analyzer（存储时生效的分词器）：用 ik_max_word，拆的更碎、索引更多，更有可能被搜出来

search_analyzer（查询时生效的分词器）：用 ik_smart，更偏向于用户想搜的分词

如果想要让 text 类型的分词字段也支持精确查询，可以创建 keyword 类型的子字段：

```
  "fields": {
    "keyword": {
      "type": "keyword",
      "ignore_above": 256 // 超过字符数则忽略查询
    }
  }
```

建表结构：

```
POST post_v1
{
  "aliases": {
    "post": {}
  },
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "analyzer": "ik_max_word",
        "search_analyzer": "ik_smart",
        "fields": {
          "keyword": {
            "type": "keyword",
            "ignore_above": 256
          }
        }
      },
      "content": {
        "type": "text",
        "analyzer": "ik_max_word",
        "search_analyzer": "ik_smart",
        "fields": {
          "keyword": {
            "type": "keyword",
            "ignore_above": 256
          }
        }
      },
      "tags": {
        "type": "keyword"
      },
      "userId": {
        "type": "keyword"
      },
      "createTime": {
        "type": "date"
      },
      "updateTime": {
        "type": "date"
      },
      "isDelete": {
        "type": "keyword"
      }
    }
  }
}
```

2、增删改查

第一种方式：ElasticsearchRepository<PostEsDTO, Long>，默认提供了简单的增删改查，多用于可预期的、相对没那么复杂的查询、自定义查询，返回结果相对简单直接。

接口代码：

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    long count();

    void deleteById(ID id);

    void delete(T entity);

    void deleteAllById(Iterable<? extends ID> ids);

    void deleteAll(Iterable<? extends T> entities);

    void deleteAll();
}
```

ES 中，_开头的字段表示系统默认字段，比如 _id，如果系统不指定，会自动生成。但是不会在 _source 字段中补充 id 的值，所以建议大家手动指定。

支持根据方法名自动生成方法，比如：

```
List<PostEsDTO> findByTitle(String title);
```

第二种方式：Spring 默认给我们提供的操作 es 的客户端对象 ElasticsearchRestTemplate，也提供了增删改查，它的增删改查更灵活，适用于更复杂的操作，返回结果更完整，但需要自己解析。

对于复杂的查询，建议用第二种方式。

三个步骤：

1. 取参数
2. 把参数组合为 ES 支持的搜索条件

3．从返回值中取结果

3、查询 DSL

参考文档：

- https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-filter-context.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-filter-context.html>
- https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl-bool-query.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/query-dsl-bool-query.html>

示例代码：

```
GET post/_search
{
  "query": {
    "bool": { // 组合条件
      "must": [ // 必须都满足
        { "match": { "title":   "鱼皮"         }}, // match 模糊查询
        { "match": { "content":   "知识星球"         }}
      ],
      "filter": [
        { "term":  { "status": "published" }}, // term 精确查询
        { "range": { "publish_date": { "gte": "2015-01-01" }}} // range 范围查
      ]
    }
  }
}
```

wildcard 模糊查询

regexp 正则匹配查询

查询结果中，score 代表匹配分数

建议先测试 DSL、再翻译成 Java

```
{
  "query": {
    "bool": {
      "must_not": [
        {
          "match": {
            "title": ""
          }
        },
      ]
      "should": [
        {
          "match": {
            "title": ""
          }
        },
        {
          "match": {
            "desc": ""
          }
        }
      ],
      "filter": [
        {
          "term": {
            "isDelete": 0
          }
        },
        {
          "term": {
            "id": 1
          }
        },
        {
          "term": {
            "tags": "java"
          }
        },
        {
          "term": {
            "tags": "框架"
          }
        }
      ],
      "minimum_should_match": 0
    }
  },
  "from": 0, // 分页
  "size": 5, // 分页
  "_source": ["name", "_createTime", "desc", "reviewStatus", "priority", "tag
```

```
  "sort": [ // 排序
    {
      "priority": {
        "order": "desc"
      }
    },
    {
      "_score": {
        "order": "desc"
      }
    },
    {
      "publishTime": {
        "order": "desc"
      }
    }
  ]
}
```

翻译为 Java：

```
        BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
        // 过滤
        boolQueryBuilder.filter(QueryBuilders.termQuery("isDelete", 0));
        if (id != null) {
            boolQueryBuilder.filter(QueryBuilders.termQuery("id", id));
        }
        if (notId != null) {
            boolQueryBuilder.mustNot(QueryBuilders.termQuery("id", notId));
        }
        if (userId != null) {
            boolQueryBuilder.filter(QueryBuilders.termQuery("userId", userId)
        }
        // 必须包含所有标签
        if (CollectionUtils.isNotEmpty(tagList)) {
            for (String tag : tagList) {
                boolQueryBuilder.filter(QueryBuilders.termQuery("tags", tag))
            }
        }
        // 包含任何一个标签即可
        if (CollectionUtils.isNotEmpty(orTagList)) {
            BoolQueryBuilder orTagBoolQueryBuilder = QueryBuilders.boolQuery(
            for (String tag : orTagList) {
                orTagBoolQueryBuilder.should(QueryBuilders.termQuery("tags",
            }
            orTagBoolQueryBuilder.minimumShouldMatch(1);
            boolQueryBuilder.filter(orTagBoolQueryBuilder);
        }
        // 按关键词检索
        if (StringUtils.isNotBlank(searchText)) {
            boolQueryBuilder.should(QueryBuilders.matchQuery("title", searchT
            boolQueryBuilder.should(QueryBuilders.matchQuery("content", searc
            boolQueryBuilder.minimumShouldMatch(1);
        }
        // 按标题检索
        if (StringUtils.isNotBlank(title)) {
            boolQueryBuilder.should(QueryBuilders.matchQuery("title", title))
            boolQueryBuilder.minimumShouldMatch(1);
        }
        // 按内容检索
        if (StringUtils.isNotBlank(content)) {
            boolQueryBuilder.should(QueryBuilders.matchQuery("content", conte
            boolQueryBuilder.minimumShouldMatch(1);
        }
        // 排序
        SortBuilder<?> sortBuilder = SortBuilders.scoreSort();
        if (StringUtils.isNotBlank(sortField)) {
            sortBuilder = SortBuilders.fieldSort(sortField);
            sortBuilder.order(CommonConstant.SORT_ORDER_ASC.equals(sortOrder)
        }
        // 分页
```

```
51          PageRequest pageRequest = PageRequest.of((int) current, (int) pageSiz
52          // 构造查询
53          NativeSearchQuery searchQuery = new NativeSearchQueryBuilder().withQu
54                  .withPageable(pageRequest).withSorts(sortBuilder).build();
55          SearchHits<PostEsDTO> searchHits = elasticsearchRestTemplate.search(s
```

动静分离设计：先模糊筛选静态数据，查出数据后，再根据查到的内容 id 去数据库查找到 **动态数据。**

数据同步

一般情况下，如果做查询搜索功能，使用 ES 来模糊搜索，但是数据是存放在数据库 MySQL 里的，所以说我们需要把 MySQL 中的数据和 ES 进行同步，保证数据一致（以 MySQL 为主）。

MySQL => ES （单向）

首次安装完 ES，把 MySQL 数据全量同步到 ES 里，写一个单次脚本

4 种方式，全量同步（首次）+ 增量同步（新数据）：

1. 定时任务，比如 1 分钟 1 次，找到 MySQL 中过去几分钟内（至少是定时周期的 2 倍）发生改变的数据，然后更新到 ES。

优点：简单易懂、占用资源少、不用引入第三方中间件

缺点：有时间差

应用场景：数据短时间内不同步影响不大、或者数据几乎不发生修改

2. 双写：写数据的时候，必须也去写 ES；更新删除数据库同理。（事务：建议先保证 MySQL 写成功，如果 ES 写失败了，可以通过定时任务 + 日志 + 告警进行检测和修复（补偿））
3. 用 Logstash 数据同步管道（一般要配合 kafka 消息队列 + beats 采集器）：
4. Canal 监听 MySQL Binlog，实时同步

Logstash

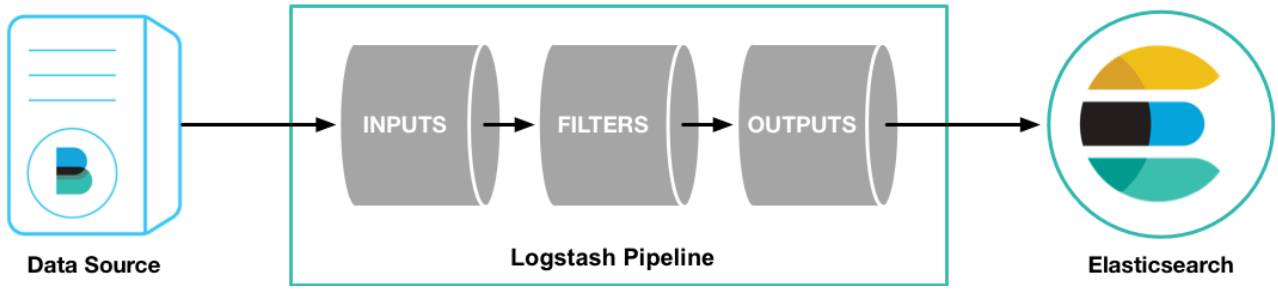
传输 和 **处理** 数据的管道

https://www.elastic.co/guide/en/logstash/7.17/getting-started-with-logstash.html <https://www.elastic.co/guide/en/logstash/7.17/getting-started-with-logstash.html>

https://artifacts.elastic.co/downloads/logstash/logstash-7.17.9-windows-x86_64.zip <https://artifacts.elastic.co/downloads/logstash/logstash-7.17.9-windows-x86_64.zip>

好处：用起来方便，插件多

缺点：成本更大、一般要配合其他组件使用（比如 kafka）



事件 Demo：

```
cd logstash-7.17.9
.\bin\logstash.bat -e "input { stdin { } } output { stdout {} }"
```

快速开始文档：https://www.elastic.co/guide/en/logstash/7.17/running-logstash-windows.html <https://www.elastic.co/guide/en/logstash/7.17/running-logstash-windows.html>

监听 udp 并输出：

```
# Sample Logstash configuration for receiving
# UDP syslog messages over port 514

input {
  udp {
    port => 514
    type => "syslog"
  }
}

output {
  stdout { codec => rubydebug }
}
```

要把 MySQL 同步给 Elasticsearch。

问题 1：找不到 mysql 的包

Error: unable to load mysql-connector-java-5.1.36-bin.jar from :jdbc_driver_library, file not readable (please check user and group permissions for the path)

Exception: LogStash::PluginLoadingError

解决：修改 Logstash 任务配置中的 jdbc_driver_library 为驱动包的绝对路径（驱动包可以从 maven 仓库中拷贝）

增量配置：是不是可以只查最新更新的？可以记录上次更新的数据时间，只查出来 > 该更新时间的数据

小知识：预编译 SQL 的优点?

1. 灵活

2．模板好懂

3．快（有缓存）

4．部分防注入

sql_last_value 是取上次查到的数据的最后一行的指定的字段，如果要全量更新，只要删除掉 E:\software\ElasticStack\logstash-7.17.9\data\plugins\inputs\jdbc\logstash_jdbc_last_run 文件即可（这个文件存储了上次同步到的数据）

```
input {
  jdbc {
    jdbc_driver_library => "E:\software\ElasticStack\logstash-7.17.9\config\m
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://localhost:3306/my_db"
    jdbc_user => "root"
    jdbc_password => "123456"
    statement => "SELECT * from post where updateTime > :sql_last_value"
    tracking_column => "updatetime"
    tracking_column_type => "timestamp"
    use_column_value => true
    parameters => { "favorite_artist" => "Beethoven" }
    schedule => "*/5 * * * * *"
    jdbc_default_timezone => "Asia/Shanghai"
  }
}

output {
  stdout { codec => rubydebug }
}

```

注意查询语句中要按 updateTime 排序，保证最后一条是最大的：

```
input {
  jdbc {
    jdbc_driver_library => "E:\software\ElasticStack\logstash-7.17.9\config\m
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://localhost:3306/my_db"
    jdbc_user => "root"
    jdbc_password => "123456"
    statement => "SELECT * from post where updateTime > :sql_last_value and u
    tracking_column => "updatetime"
    tracking_column_type => "timestamp"
    use_column_value => true
    parameters => { "favorite_artist" => "Beethoven" }
    schedule => "*/5 * * * * *"
    jdbc_default_timezone => "Asia/Shanghai"
  }
}

output {
  stdout { codec => rubydebug }

  elasticsearch {
    hosts => "http://localhost:9200"
    index => "post_v1"
    document_id => "%{id}"
  }
}
```

两个问题：

1. 字段全变成小写了
2. 多了一些我们不想同步的字段

可以编写过滤：

```
input {
  jdbc {
    jdbc_driver_library => "E:\software\ElasticStack\logstash-7.17.9\config\m
    jdbc_driver_class => "com.mysql.jdbc.Driver"
    jdbc_connection_string => "jdbc:mysql://localhost:3306/my_db"
    jdbc_user => "root"
    jdbc_password => "123456"
    statement => "SELECT * from post where updateTime > :sql_last_value and u
    tracking_column => "updatetime"
    tracking_column_type => "timestamp"
    use_column_value => true
    parameters => { "favorite_artist" => "Beethoven" }
    schedule => "*/5 * * * * *"
    jdbc_default_timezone => "Asia/Shanghai"
  }
}

filter {
    mutate {
        rename => {
          "updatetime" => "updateTime"
          "userid" => "userId"
          "createtime" => "createTime"
          "isdelete" => "isDelete"
        }
        remove_field => ["thumbnum", "favournum"]
    }
}

output {
  stdout { codec => rubydebug }

  elasticsearch {
    hosts => "127.0.0.1:9200"
    index => "post_v1"
    document_id => "%{id}"
  }
}
```

订阅数据库流水的同步方式 Canal

https://github.com/alibaba/canal/ <https://github.com/alibaba/canal/>

优点：实时同步，实时性非常强

原理：数据库每次修改时，会修改 binlog 文件，只要监听该文件的修改，就能第一时间得到消息并处理

canal：帮你监听 binlog，并解析 binlog 为你可以理解的内容。

它伪装成了 MySQL 的从节点，获取主节点给的 binlog，如图：

快速开始：https://github.com/alibaba/canal/wiki/QuickStart
<https://github.com/alibaba/canal/wiki/QuickStart>

windows 系统，找到你本地的 mysql 安装目录，在根目录下新建 my.ini 文件：

```
[mysqld]
log-bin=mysql-bin # 开启 binlog
binlog-format=ROW # 选择 ROW 模式
server_id=1 # 配置 MySQL replaction 需要定义，不要和 canal 的 slaveId 重复
```

如果 java 找不到，修改 startup.bat 脚本为你自己的 java home：

```

set JAVA_HOME=C:\Users\59278\.jdks\corretto-1.8.0_302
echo %JAVA_HOME%
set PATH=%JAVA_HOME%\bin;%PATH%
echo %PATH%

```

问题：mysql 无法链接，Caused by: java.io.IOException: caching_sha2_password Auth failed

解决方案：

https://github.com/alibaba/canal/issues/3902
<https://github.com/alibaba/canal/issues/3902>

ALTER USER 'canal'@'%' IDENTIFIED WITH mysql_native_password BY 'canal';

ALTER USER 'canal'@'%' IDENTIFIED BY 'canal' PASSWORD EXPIRE NEVER;

FLUSH PRIVILEGES;

配置 kibana 可视化看板

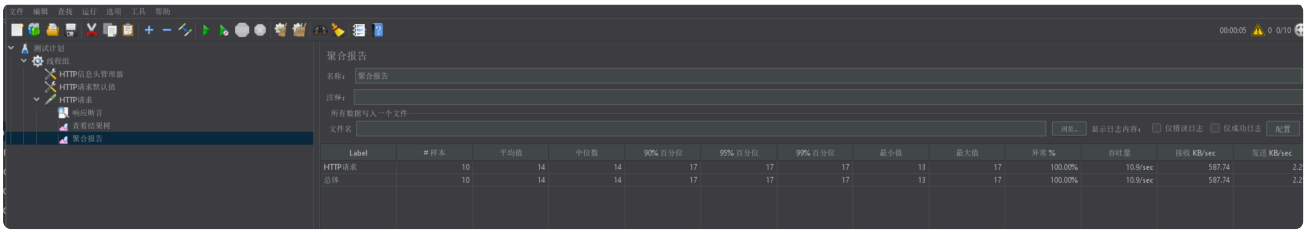
1. 创建索引
2. 导入数据
3. 创建索引模式
4. 选择图表、拖拉拽
5. 保存

压力测试

官方文档：https://jmeter.apache.org/ <https://jmeter.apache.org/>

找到 jar 包：apache-jmeter-5.5\apache-jmeter-5.5\bin\ApacheJMeter.jar 启动

配置线程组 => 请求头 => 默认请求 => 单个请求 => 响应断言 => 聚合报告 / 结果树



99%分位：99%的用户都在这个响应时间内

吞吐量：每秒处理的请求数 qps

更多学习

插件：https://jmeter-plugins.org/install/Install/ <https://jmeter-plugins.org/install/Install/>

下载后文件为plugins-manager.jar <https://jmeter-plugins.org/get/> 格式，将其放入jmeter安装目录下的lib/ext目录，然后重启jmeter，即可。

参考文章：https://blog.csdn.net/weixin_45189665/article/details/125278218 <https://blog.csdn.net/weixin_45189665/article/details/125278218>

搜索建议

参考官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/7.17/search-suggesters.html <https://www.elastic.co/guide/en/elasticsearch/reference/7.17/search-suggesters.html>

示例：

```
GET post_v1/_search
{
  "query": {
    "match": { "content": "鱼皮不欠费" }
  },
  "highlight": {
    "fields": {
      "content": {
        "pre_tags" : ["<h1>"], "post_tags" : ["</h1>"]
      }
    }
  },
 "suggest" : {
    "my-suggestion" : {
      "text" : "鱼皮不欠费",
      "term" : {
        "field" : "content"
      }
    }
  }
}
```

搜索高亮

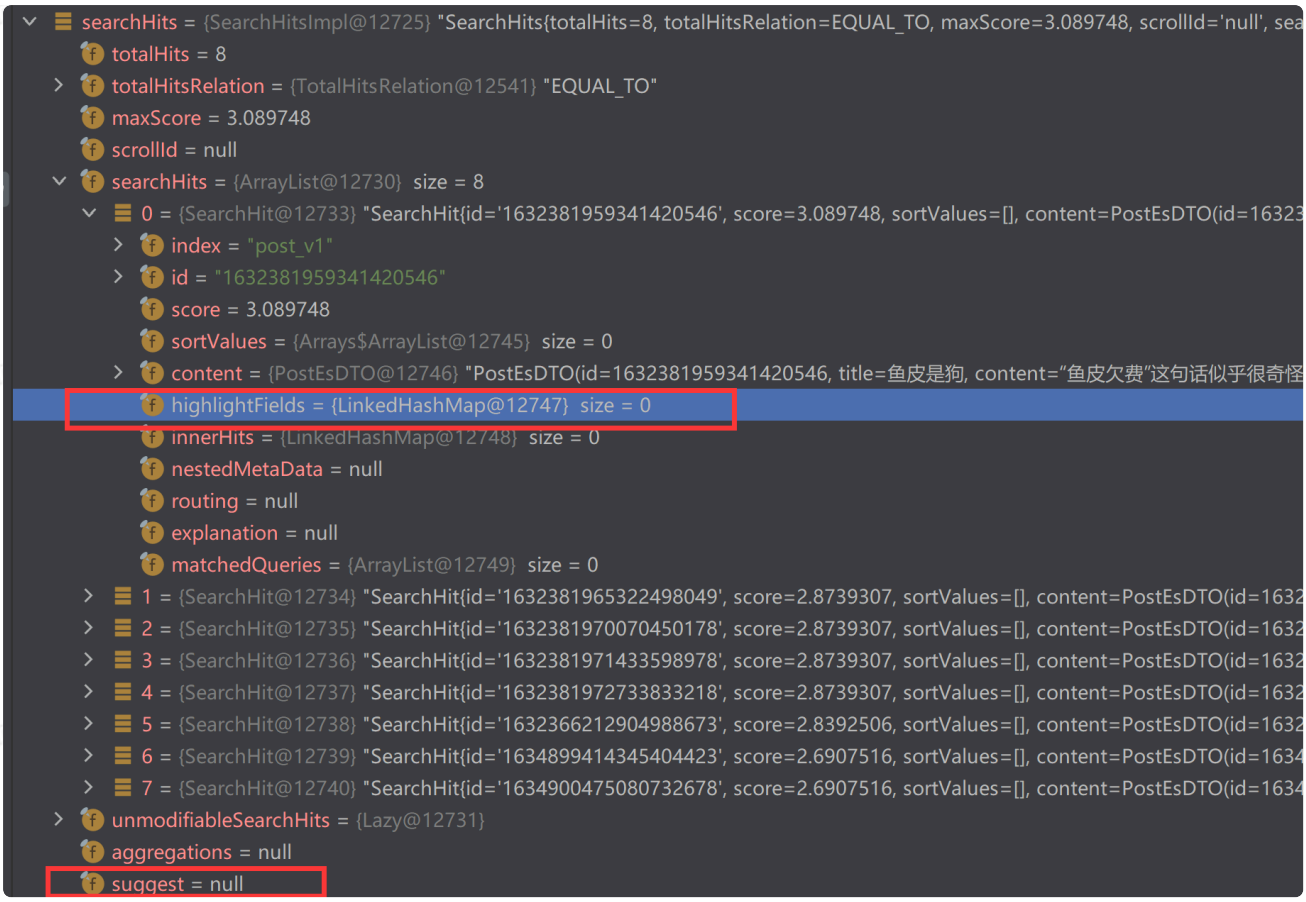
参考官方文档：

https://www.elastic.co/guide/en/elasticsearch/reference/7.17/highlighting.html
<https://www.elastic.co/guide/en/elasticsearch/reference/7.17/highlighting.html>

示例：

```
GET post_v1/_search
{
  "query": {
    "match": { "content": "鱼皮欠费" }
  },
  "highlight": {
    "fields": {
      "content": {
        "pre_tags" : ["<h1>"], "post_tags" : ["</h1>"]
      }
    }
  }
}
```

高亮、建议都可以从返回值里拿到：



前端防抖节流（自行扩展）

问题：用户频繁输入、频繁点搜索按钮怎么办？

解决：使用 lodash 工具库实现防抖和节流。

节流：每段时间最多执行 x 次（比如服务器限流）
https://www.lodashjs.com/docs/lodash.throttle
<https://www.lodashjs.com/docs/lodash.throttle>

防抖：等待一段时间内没有其他操作了，才执行操作（比如输入搜索）
https://www.lodashjs.com/docs/lodash.debounce
<https://www.lodashjs.com/docs/lodash.debounce>

接口稳定性优化（自行扩展）

问题：调用第三方接口不稳定怎么办？（比如 bing 接口）

使用 guava-retrying 库实现自动重试：https://github.com/rholder/guava-retrying
<https://github.com/rholder/guava-retrying>

可以阅读鱼皮原创的这篇文章学习：https://cloud.tencent.com/developer/article/1752086 <https://cloud.tencent.com/developer/article/1752086>