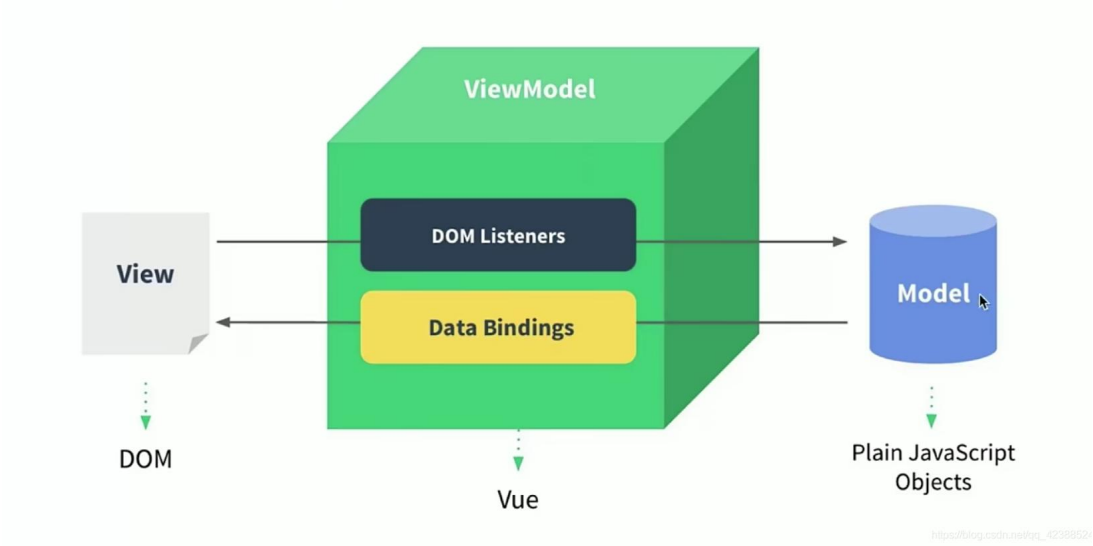
TopMat 项目技术文档

一、项目技术构建

1、技术框架设计

1) MVVM 模型

MVVM 是 Model-View-ViewModel 的简写，即模型-视图-视图模型。它本质上就是 MVC 的改进版。模型指的是后端传递的数据；视图指的是所看到的页面。视图模型是 MVVM 模式的核心，它是连接 View 和 Model 的桥梁。它有两个方向：一是将模型转化成视图，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。二是将视图转化成模型，即将所看到的页面转化成后端的数据。



2) 前后端分离

本项目采用前后端分离的开发模式，有效地对前端和后端的开发进行解耦，并为以后多端化服务打下基础。前后端分离的核心思想就是前端框架或工具通过 http 和 websocket 等协议调用后端接口，并通过字符串等格式的数据进行交互。

3) 轻量化 WEB 服务框架

本项目使用轻量化 web 服务框架作为后端，用于对前端数据的处理，以及与数据库的交互。轻量化服务框架构建简单，没有复杂的运行逻辑，具有很强的扩展性，非常符合项目所需。

本项目主要使用如下技术栈：

前端技术	
后端技术	
数据库	
部署	

2、前端技术栈

前端采用如下技术栈进行页面开发

Option	Selected
编程语言	Javascript
框架	Vue.js
网络通信	axios
UI 组件库	BootstrapVue

1) Vue.js

本项目前端搭建基本完全围绕 Vue 2.0 来进行，由于 Vue 只关注视图层，是一个构建数据的视图集合，所以大小只有几十 kb，是非常轻便的框架。同时 Vue 通过简洁的 API 提供高效的数据绑定和灵活的组件系统，开发起来更便捷。并且，很重要的一点，Vue 的生态系统在近几年市场上越来越丰富和完备，在国内相较于 React 框架的生态明显更好，可以在满足开发与应用的各式需求的同时还能带来更加丰富的交互体验。

2) Axios

本项目使用 axios 来完成 ajax 请求，axios 实际是通过 promise 实现对 ajax 技术的一种封装，符合最新的 ES 规范，既支持后端也支持浏览器端。使用 promise 管理异步后，就可以告别传统的 callback 方式，不再存在因各个接口的依赖关系

产生的层层嵌套问题。并且 axios 还有非常丰富的配置项，支持拦截器等高级配置。

3) BootStrapVue

基于 Vue.js 和 Bootstrap v4 的前端 CSS 框架，具有响应式布局、移动优先、按需组装、交互友好、可自由配置、免费开放等优势。

3、后端技术栈

后端采用如下技术栈进行开发

Option	Selected
编程语言	Python
框架	Flask
接口测试	Postman
数据库	MongoDB

1) Flask

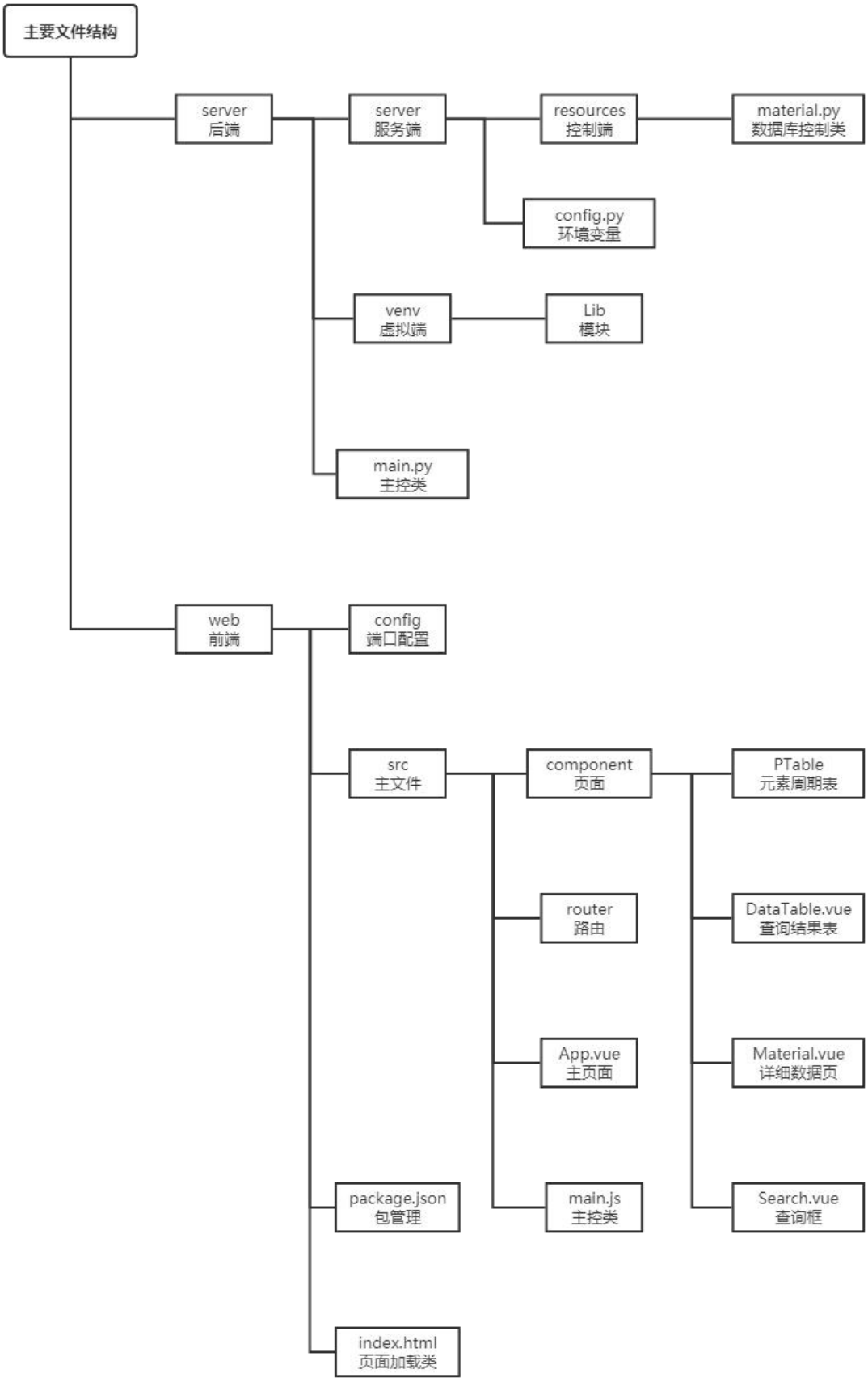
Flask 框架核心简单，同时在使用过程同样可以保持功能的丰富与扩展性，主要包含 Werkzeug 和 jinja2 两个核心函数库，这些基础函数为 Web 项目开发过程提供了丰富的基础组件。Flask 中的 jinja2 模板引擎，提高前端代码的复用率，可以大大提高开发效率并且有利于后期的开发与维护。Flask 不会指定数据库和模板引擎等对象，用户可以根据需要自己选择各种数据库。Flask 不提供表单功能验证，在项目实施过程中可以自由配置，从而为应用程序开发提供数据库抽象层基础组件，支持进行表单数据合法性验证、文件上传处理、用户身份认证和数据库集成等功能。

2) Postman

Postman 是一个 Chrome 扩展，提供功能强大的 Web API & HTTP 请求调试。它能够发送任何类型的 HTTP 请求 (GET, HEAD, POST, PUT..)，附带任何数量的参数+ headers。支持不同的认证机制，接收到的响应语法高亮（HTML，JSON 或 XML）。Postman 能够保留了历史的请求，这样可以很容易地重新发送请求，有一个“集合”功能，用于存储所有相同的请求。

二、项目代码结构

代码文件主要结构如下：



三、前端代码详解

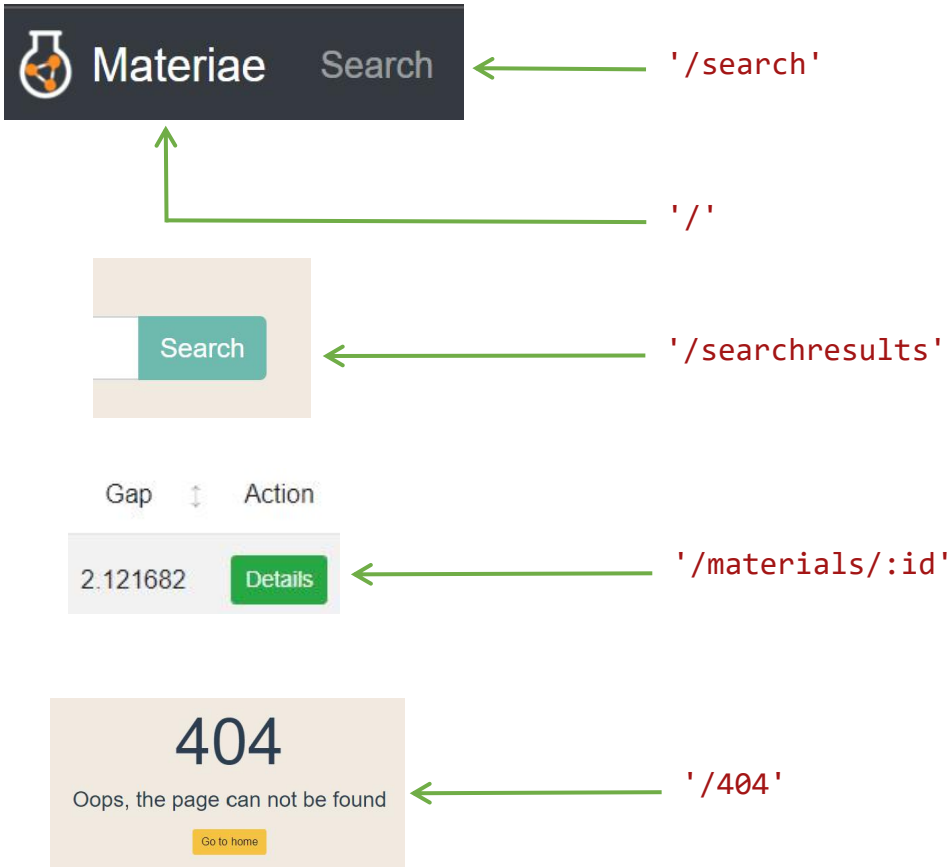
1、前端路由

前端路由文件(web\src\router\index.js)

index.js

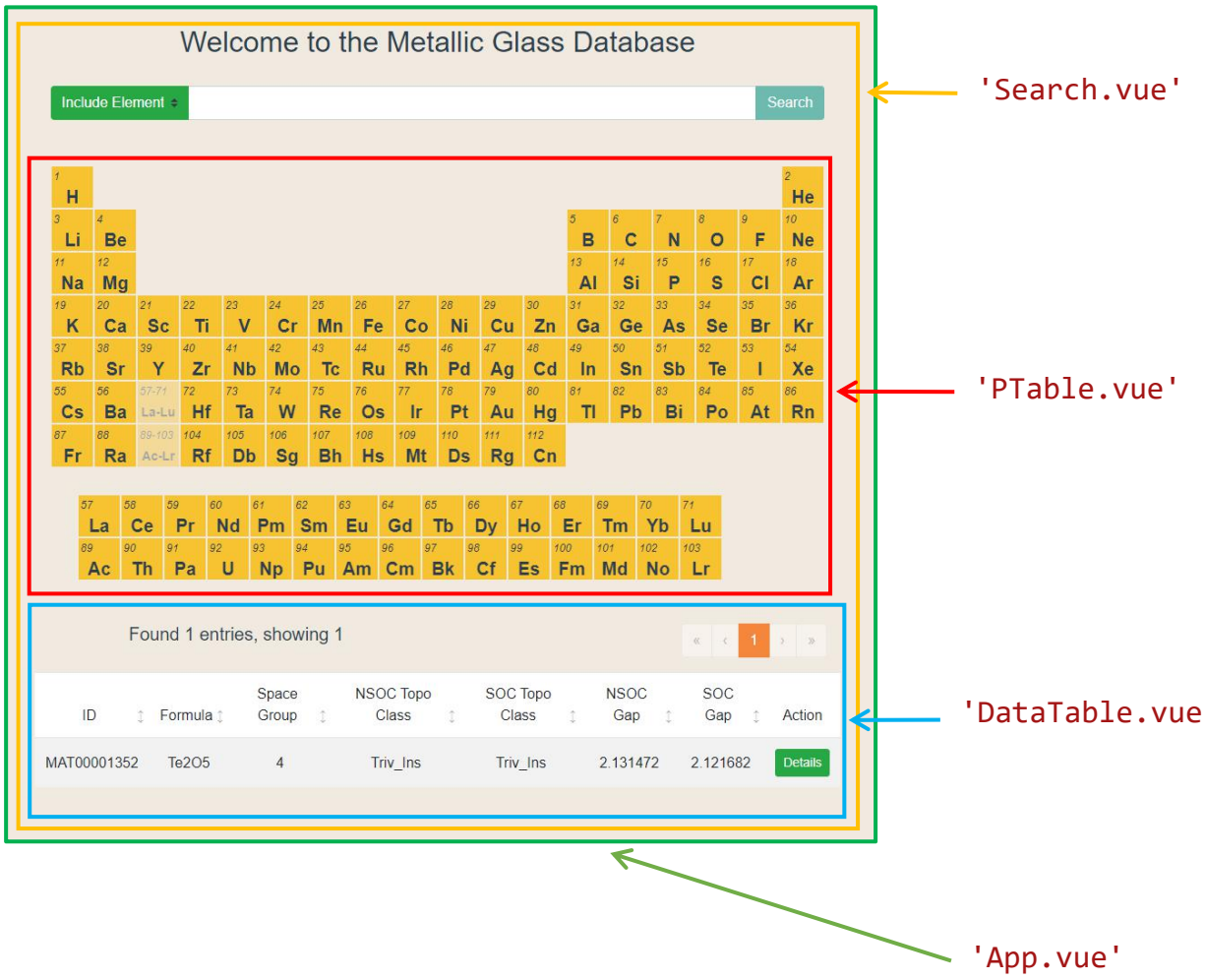
```
//路由文件模板
//页面位置
import materials from '@/components/Material'

......
//单个路由
{
    path: '/materials/:id',  //URL
    name: 'materials',  //渲染组件名
    component: materials  //对应页面位置
}

......
```

目前一共有 5 个 URL，它们分别是：

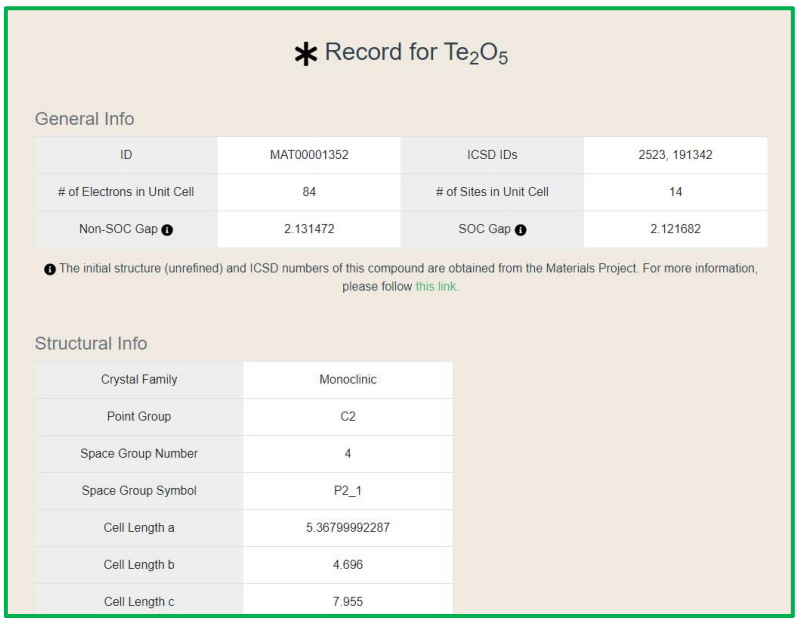


2、前端页面布局

1) 页面及其对应文件的关系如下：



'Material.vue'



2) 页面布局代码结构：

App.vue

```
//代码结构
<template>
   <b-navbar toggleable="md"type="dark"variant="dark"class="w-100">
      ......    //导航栏
   </b-navbar>
   <b-container fluid class="flex-grow-1 px-0 d-flex">
      <keep-alive :include="['Search']">  //指向 Search.vue，在路由中设置
         ......   //页面主要部分
      </keep-alive>
   </b-container>
   <b-container fluid class="page-footer text-light px-5 pt-3">
      ......  //页脚
   </b-container>
</template>
```

Search.vue

```
//代码结构
<template>
   <b-container fluid class="d-flex align-items-stretch">
      .....
         <b-input-group class="search-bar"> //搜索框
         ......
         </b-input-group>
         ......
         <PTable full class="mx-auto mb-5" :value="elems"
         @input="handlePTInput"></PTable>
         .....
         <DataTable :data="data"></DataTable>
         .....
   </b-container>
</template>
<script>
   //引用其他页面文件并命名
   import PTable from './PTable/PTable.vue'
   import DataTable from './DataTable.vue'
   .....
   components: {
      DataTable,
      PTable
   },
   .....
</script>
```

DataTable.vue

```
//代码结构
<template>
    .....
    <b-table striped
    hover :fields="fields" :per-page="pageSize" :items="tableData" :current
    -page="page">  //数据表格
        <template slot="action" slot-scope="row">
            <b-button size="sm" variant="success" :to="'/materials/' +
            row.item.id"> //按钮，点击后跳转'/materials/:id'
                Details    //id 即为该行数据的 id
            </b-button>
        </template>
    </b-table>
</template>
<script>
    data () {
        return {
            fields: [
                //表格数据
                {key: 'id', label: 'ID', sortable: true},
                //key 代表数据名称，label 代表标签，sortable 代表是否排序
                .....
            ],
        }
    },
    .....
</script>
```

3、前端数据传输

1) DataTable 数据接收：

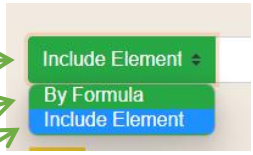
DataTable.vue

```
//部分代码内容
<script>
    .....
    computed: {
        tableData () {
            this.data.materials.forEach((mat, i) => { //从后端接收 data 数据
                rtn.push({
                    'id': mat.id, //将 data 数据中的 id 项，命名为 id，对应 key 名称
                    .....
                })
            })
        },
</script>
```

2) 搜索数据：

Search.vue

```
//部分代码内容
<template>
    .....
    <b-btn @click="handleSearch" variant="info" slot="append"
    class="text-white">Search</b-btn> //搜索按钮，控制类为 handleSearch ①
</template>
<script>
function getDefaultData () {
    .....
    searchMode: 'elements',
    modeOptions: [
        {value: 'formula', text: 'By Formula'},
        {value: 'elements', text: 'Include Element'}
    ],
}
    .....
    computed: {
        query () { //输入数据预处理 ③
            let query = {}
            if (this.searchMode === 'formula' && this.elems.length) {
                query['formula'] = this.elems.join('')
            } else if (this.searchMode === 'elements' && this.elems.length) {
                query['elements'] = this.elems.join('')
            }
            return query //返回处理好的数据
        }
    },
    .....
    handleSearch () { //按钮控制类 ②
        this.queryServer(this.query) //将输入的数据发给 query()进行预处理
        this.$router.push({name: 'searchResults', query: this.query})
        //收到处理好的数据后，跳转页面
    },
    .....
    queryServer (query) { //将处理好的数据发送给后端
        this.axios.get('api/materials/search', { params: query })
            .....        //后端的 URL
    }
</script>
```



四、后端代码详解

1、后端路由

后端路由文件(server\server\main.py)

main.py

```
//路由文件模板
//数据库连接
client = MongoClient(config.DB_URI)
db = client.topmat
connect(config.DB_URI)
......
//路由
from resources import material //路由文件位置(/resources/material)
//后端 URL 与其对应文件中的 class 类
api.add_resource(material.Materials, '/materials/search')
api.add_resource(material.MaterialDetail, '/material')
......
```

2、后端数据传输

main.py

```
//部分代码内容
//class 类名与 main.py 中相对应
class Materials(Resource):
def _get(self):
        //设置数据结构为字典
        query = defaultdict(dict)
        //将前端传过来的数据处理成字典
        for key in request.args:
            value = request.args[key]
            query[key] = value
        //处理好的数据写入 mongo 语句进行查找
        materials = db.materials.find(query).limit(30000)
        //将 mongo 查找出的数据处理成 JSON
        materials = [filter(mat, self.return_keys) for mat in materials]
        //数据传给前端
         return Response(encoder.encode({'count': len(materials),
        'materials': materials}))
```

将前端数据处理成字典，是因为 mongo 语句查找格式与字典类似，相关语句可以直接插入 mongo 语句中

mongo 查找的结果也为字典型，前端所用的 JavaScript 并不能直接处理字典型数据，将查找的数据处理成 JSON 能使前端更方便的处理后端数据