Relatório Detalhado sobre os Códigos Python - Mecanismo de Busca com Whoosh e Elastic Search

Lucas de Oliveira Pereira Braga[1]

[1]Departamento de Ciência da Computação– Universidade Federal Rural do Rio de Janeiro (UFRRJ)

Abstract: *This detailed report provides an in-depth analysis of two Python search engine implementations: one using Whoosh and the other using ElasticSearch. The Whoosh-based system, encapsulated in the `SearchEngine` class, employs in-memory indexing and JSON serialization for query processing. It demonstrates good coding practices but lacks attention to SSL certificate security. In contrast, the ElasticSearch-based system, also encapsulated in the `SearchEngine` class, interacts with a remote ElasticSearch cluster. While addressing SSL certificate concerns, it introduces complexity and configuration requirements. The report concludes with a comprehensive comparison, highlighting performance, scalability, security considerations, and recommendations for each search engine.*

Resumo: *Este relatório detalhado oferece uma análise aprofundada de duas implementações de mecanismos de busca em Python: uma utilizando Whoosh e a outra utilizando ElasticSearch. O sistema baseado em Whoosh, encapsulado na classe `SearchEngine`, utiliza indexação em memória e serialização JSON para processamento de consultas. Demonstrando boas práticas de programação, o código carece de atenção à segurança de certificados SSL. Em contraste, o sistema baseado em ElasticSearch, também encapsulado na classe `SearchEngine`, interage com um cluster remoto do ElasticSearch. Ao abordar preocupações com certificados SSL, introduz complexidade e requisitos de configuração. O relatório conclui com uma comparação abrangente, destacando desempenho, escalabilidade, considerações de segurança e recomendações para cada mecanismo de busca.*

Parte 1 :Whoosh

1. Componentes Principais

1.1. Estrutura do Código

O código está organizado em uma classe principal, **`SearchEngine`**, que encapsula a lógica do mecanismo de busca. As funcionalidades principais incluem a inicialização do índice, a indexação de documentos, a realização de consultas e a obtenção do tamanho do índice.

1.1.1. Importações

O código utiliza diversas bibliotecas essenciais, incluindo **`json`** para manipulação de dados JSON e bibliotecas da Whoosh para a criação do mecanismo de busca.

1.1.1. Importações:

```
from typing import Dict, List, Sequence
from whoosh.index import create_in
from whoosh.fields import *
from whoosh.qparser import MultifieldParser
from whoosh.filedb.filestore import RamStorage
from whoosh.analysis import StemmingAnalyzer
import json
```

1.2. Utilização do Formato JSON

A serialização e desserialização de documentos são realizadas utilizando o formato JSON. O campo 'raw' é adicionado aos documentos para armazenar uma versão serializada dos mesmos.

```
d['raw'] = json.dumps(doc)
```

2. Indexação de Consultas

2.1. Processamento de Consultas do Arquivo "cran.qry"

As consultas são lidas do arquivo "cran.qry" e processadas para criar uma lista de dicionários representando cada consulta.

```
if __name__ == '__main__':
    with open('cran.qry', 'r') as f:
        queries = f.read().split('.I')[1:]
    query_docs = []
    current_query = {}
    for query in queries:
        lines = query.strip().split('\n')
        current_query['id'] = lines[0].strip()
        current_query['query'] = ' '.join(lines[2:]).strip()
        query_docs.append(current_query.copy())
```

3. Esquema Whoosh

3.1. Definição do Esquema

O esquema Whoosh define a estrutura dos documentos a serem indexados, incluindo campos como 'id' e 'query'. O campo 'query' é analisado utilizando um **StemmingAnalyzer** para processamento de texto.

```
schema = Schema(
        id=ID(stored=True),
        query=TEXT(stored=True, analyzer=StemmingAnalyzer())
    )
```

4. Execução Principal

4.1. Indexação e Consulta

O código principal instancia o **SearchEngine**, indexa as consultas processadas e realiza consultas usando o próprio arquivo "cran.qry".

```
schema = Schema(
        id=ID(stored=True),
        query=TEXT(stored=True, analyzer=StemmingAnalyzer())
    )
    engine = SearchEngine(schema)
    engine.index_documents(query_docs)
    print(f"indexed {engine.get_index_size()} queries")
    fields_to_search = ["query"]s
    for q in query_docs:
```

4.1. Continuação:

```
        print(f"Query:: {q['query']}")
        print("\t", engine.query(q['query'], fields_to_search,
highlight=True))
        print("-" * 70)
```

5. Recomendações e Observações Adicionais

5.1. Segurança

- **Certificado SSL:** A desativação da verificação de certificado SSL (**verify_certs=False**) é uma prática insegura. Recomenda-se o uso de um certificado SSL válido para ambientes de produção.

5.2. Manutenção e Melhorias

- **Encapsulamento:** Considere encapsular a lógica de leitura do arquivo e processamento de consultas em métodos da classe para melhor organização e reutilização do código.
- **Atualizações no Índice:** Expanda a lógica para permitir a adição de novos documentos ou atualizações nos documentos existentes.

5.3. Documentação e Comentários

- **Documentação Adequada:** Adicione comentários e documentação adequada para melhorar a compreensão do código, especialmente para métodos e lógicas mais complexas.

5.4. Melhorias de Funcionalidades

- **Logging:** Introduza capacidade de logging para facilitar a depuração e monitoramento.
- **Testes Unitários:** Considere a inclusão de testes unitários para verificar o comportamento esperado do código.

5.5. Boas Práticas de Desenvolvimento

- **Integração Contínua e Controle de Versão:** Utilize práticas de integração contínua e controle de versão para facilitar o desenvolvimento colaborativo e garantir a qualidade do código.

Parte 2: Elastic Search

1.Estrutura do Código

1.1 Classe SearchEngine

A classe **SearchEngine** encapsula as funcionalidades do mecanismo de busca com ElasticSearch. Durante a inicialização, são configuradas as informações do cluster, como URL, autenticação básica e desativação das verificações de certificados SSL.

```
class SearchEngine:

    def __init__(self, esquema):
        self.esquema = esquema
        self.index_name = 'my_index'
        self.es = Elasticsearch(
            ["https://localhost:9200"],
            basic_auth=("elastic", "NxdaBhjiQg7NrB*DmgT4"),
            verify_certs=False
)
        if not self.es.indices.exists(index=self.index_name, ):
            self.es.indices.create(index=self.index_name,
body={"mappings": {"properties": self.esquema}})
```

1.2 Indexação de Documentos (**indexar_documentos**)

A função **indexar_documentos** recebe uma sequência de documentos e os indexa no cluster ElasticSearch utilizando a biblioteca **elasticsearch**.

```
 def indexar_documentos(self, documentos: Sequence):
        for documento in documentos:
            self.es.index(index=self.index_name, body=documento)
```

1.3 Obter Tamanho do Índice (`obter_tamanho_do_indice`)

A função **`obter_tamanho_do_indice`** utiliza a API Cat do ElasticSearch para obter o número de documentos no índice.

```
def obter_tamanho_do_indice(self) -> int:
        response = self.es.cat.count(index=self.index_name, h='count')
        return int(response.strip())
```

1.4 Consulta (`consulta`)

A função **`consulta`** realiza consultas no índice ElasticSearch utilizando uma abordagem multi_match para procurar termos em diferentes campos especificados.

```
if __name__ == '__main__':
    esquema = {
        'id': {'type': 'integer'},
        'title': {'type': 'text'},
        'author': {'type': 'text'},
        'bibliography': {'type': 'text'},
        'content': {'type': 'text'},
    }

    mecanismo = SearchEngine(esquema)
    caminho_dataset = 'cran.all.1400'
    colecao = ler_colecao(caminho_dataset)
    mecanismo.indexar_documentos(colecao)

    print(f"Indexed {mecanismo.obter_tamanho_do_indice()} documents")

    campos_a_pesquisar = ["title", "author", "bibliography", "content"]

    # Ler as queries do arquivo cran.qry
    queries = ler_queries('cran.qry')
    i = 0
    # Utilizar as queries do arquivo cran.qry para as consultas
    for q in queries:
        i = i + 1
        print(f"Query:: {q['query']}")
        print("\t", mecanismo.consulta(q['query'], campos_a_pesquisar, destaque=True))
        print("-" * 70)
    print(f"Total de queries: {i}")
```

2. Leitura da Coleção e Consultas

O código principal lê a coleção de documentos do arquivo "cran.all.1400", instancia o **`SearchEngine`**, indexa os documentos e realiza consultas.

```
if __name__ == '__main__':
```

```
    esquema = {'id': {'type': 'integer'},'title': {'type':
'text'},'author': {'type': 'text'},'bibliography': {'type':
'text'},'content': {'type': 'text'},}
```

2. Leitura da Coleção e Consultas:

```
    mecanismo = SearchEngine(esquema)
    caminho_dataset = 'cran.all.1400'

    colecao = ler_colecao(caminho_dataset)
    mecanismo.indexar_documentos(colecao)
    print(f"Indexed {mecanismo.obter_tamanho_do_indice()} documents")
    campos_a_pesquisar = ["title", "author", "bibliography", "content"]
    queries = ler_queries('cran.qry')
    for q in queries:
        print(f"Query:: {q['query']}")
        print("\t", mecanismo.consulta(q['query'], campos_a_pesquisar,
        destaque=True))
        print("-" * 70)
    print(f"Total de queries: {i}")
```

Parte 3: Comparação entre Mecanismos de Busca: Whoosh e ElasticSearch

1. Mecanismo de Busca com Whoosh

1.1 Estrutura e Funcionalidades

O mecanismo de busca com Whoosh utiliza a biblioteca Whoosh para criar um índice de documentos em memória. A classe **SearchEngine** encapsula as operações principais, incluindo indexação, consulta e obtenção do tamanho do índice.

1.2 Boas Práticas e Segurança

O código segue boas práticas de desenvolvimento, utiliza a serialização em formato JSON para preservar os documentos originais e inclui um esquema de análise de texto. No entanto, não aborda considerações de segurança, como verificação de certificados SSL.

2. Mecanismo de Busca com ElasticSearch

2.1 Estrutura e Funcionalidades

O mecanismo de busca com ElasticSearch utiliza a biblioteca **elasticsearch** para interagir com um cluster ElasticSearch remoto. A classe **SearchEngine** abrange operações como indexação, consulta e obtenção do tamanho do índice.

2.2 Considerações de Segurança

Ao contrário do código Whoosh, o código ElasticSearch aborda a segurança desativando as verificações de certificados SSL. Embora seja uma prática insegura, é comum em ambientes de desenvolvimento.

3. Comparação Geral

3.1 Desempenho e Escalabilidade

- Whoosh opera em memória e é adequado para índices menores.
- ElasticSearch é escalável para grandes volumes de dados distribuídos.

3.2 Complexidade e Configuração

- Whoosh é simples de configurar e usar, ideal para pequenos projetos.
- ElasticSearch requer configuração de um cluster e é mais complexo, porém, oferece recursos avançados.

3.3 Segurança

- Ambos os códigos desativam verificações de certificado SSL, mas é mais crítico no ElasticSearch, que frequentemente opera em ambientes de produção.

3.4 Escolha entre Whoosh e ElasticSearch

- Whoosh é adequado para projetos menores e simples.
- ElasticSearch é preferível para projetos maiores, escaláveis e ambientes de produção, mas requer configuração e considerações de segurança.

4. Conclusão Final

Ambos os mecanismos de busca têm suas vantagens e desvantagens. A escolha entre Whoosh e ElasticSearch depende das necessidades específicas do projeto, do tamanho do índice e dos requisitos de escalabilidade e segurança. O Whoosh é fácil de começar, enquanto o ElasticSearch oferece uma solução robusta para aplicações complexas.

UFRRJ