Recuperação da Informação - Relatório de Arquitetura de Máquina de Busca

Daniel Gunna Santana da Silva Souza
Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática - ICEI
Ciência da Computação
daniel.gunna@sga.pucminas.br

Marcelo Rodrigues dos Santos Junior
Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática - ICEI
Ciência da Computação
marrsantosjunior@gmail.com

Abstract

O objetivo deste relatório é definir e apresentar a arquitetura e os componentes de uma máquina de busca vertical, que será capaz de identificar entidades, desenvolvida na disciplina de Recuperação da Informação. Serão apresentados os componentes da arquitetura da máquina de busca, introduzindo conceitos gerais sobre o funcionamento de cada um deles, assim como a justificativa e objetivo de cada um destes componentes, levando-se em consideração o contexto de aplicação.

1. Introdução

Este documento descreve a arquitetura de uma máquina de busca, definindo os principais conceitos necessários para isto. O documento está estruturado em tópicos, onde o tópico inicial descreverá o objetivo da definição da arquitetura da máquina de busca. Os tópicos seguintes serão um para cada um dos componentes mais gerais da arquitetura da máquina de busca: coletor, indexador e processador de consultas. Em cada um destes tópicos estes componentes serão apresentados de maneira mais aprofundada, definindo os sub-componentes que irão compor os componentes mais gerais e justificando sua inclusão na arquitetura e sua importância na resolução do problema proposto bem como detalhes de sua implementação. Em cada tópico serão apresentados também o funcionamento dos componentes em conjunto através de imagens e diagramas de componentes. No tópico seguinte uma síntese sobre o funcionamento completo da máquina de busca será apresentada, e no tópico final as referências bibliográficas.

2. Objetivo

Muito dos documentos encontrados na Web fazem referências aos mais diversos tipos de entidades como pessoas, lugares, empresas, instituições etc. Por isso uma máquina de busca que seja capaz de recuperar estes documentos, identificando e classificando estas entidades se faz necessária, para prover resultados mais satisfatórios as buscas realizadas pelo usuário. Esta questão é pertinente também, para realizar buscas que gerem resultados que levem em consideração a associação entre conceitos e entidades, provendo ao usuário não apenas uma ranking contendo os documentos que são referenciados por sua consulta, mas também pequenos trechos mais complexos de informação com maior valor semântico, apresentados através de uma maneira mais amigável utilizando infoboxes, infográficos, imagens associadas a consulta etc.Para isto deve-se definir uma arquitetura com componentes que sejam capazes de prover estas funcionalidades de maneira eficiente tanto do ponto de vista do usuário quanto a aspectos técnicos.

3. Coletor

O coletor é o componente da máquina de busca responsável por coletar e armazenar em uma base de dados os documentos da Web da maneira mais eficiente possível, para posterior utilização nos processos de indexação e processamento de consulta. Para isto o coletor realiza requisições web, a partir de uma lista de URL's semente, para os servidores onde os documentos se encontram, e realiza o download dos mesmos e os armazena na base de dados. Logo após os documentos são processados e passam pelo processo de parceamento, onde o texto é extraído e as novas URL's contidas nestes documentos são extraídas e usadas para realimentar o coletor. Como a máquina de busca deve ser capaz de identificar entidades, alguns sub-componentes devem prover maneiras de identificá-las ou facilitar esta identificação durante a coleta de documentos. Mais detalhes a cerca disto serão esclarecidos nos sub-tópicos desta seção. Na Figura 1 encontramos o diagrama dos componentes ilustrando a arquitetura e o funcionamento do coletor. Logo após isto cada item da arquitetura e seu funcionamento serão explicados de maneira mais detalhada nos sub-tópicos a seguir.

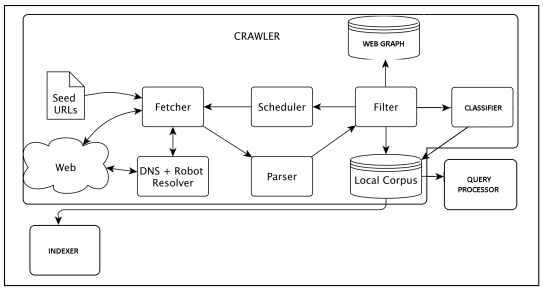

Figura 1. Diagrama dos Componentes da Arquitetura do Coletor da Máquina de Busca.Adaptado de [BRANDÃO]

3.1. Detalhes Gerais da Implementaç[a]o

O coletor foi implementado utilizando a linguagem Java, na versão 8 do seu kit de desenvolvimento.A IDE utilizada, foi a IDE Eclipse Neon para desenvolvimento Java Enterprise Edition.

3.2. DNS e Robot Resolver

Este componente é acionado antes do coletor realizar a requisição pelo download do documento no servidor, pois ele é responsável pela resolução das URL's para os respectivos endereços IP dos servidores onde os documentos serão obtidos. Este componente também é responsável por considerar durante o processo de coleta, políticas de boas maneiras com relação aos servidores e as restrições que eles impões sobre o download, coleta, indexação etc. de documentos neles hospedados, políticas estas definidas através de um arquivo que se encontra no diretório raiz ou em cada sub-diretório do servidor, conhecido como *robots.txt*. Neste processo também pode ocorrer o armazenamento dos endereços IP dos servidores e das políticas de boas maneiras, criando assim uma estrutura de *cache* que poderá ser acessada eficientemente em consultas posteriores.

3.2.1 Implementação do componente

Este componente foi implementado no arquivo ***DnsResolver.java***. A função principal desta classe é a função *String resolveAddress(Url)*, esta função resolve uma URL passada como parâmetro e obtém o arquivo robots.txt da URL. Logo após isso o endereço e o robots.txt são armazenados em um cache na memória principal, com tamanho limitado, que posteriormente é esvaziado quando este tamanho máximo é atingido.Esta estratégia foi abordada na construção do cache, pois observando a coleta e a implementação, percebeu-se que a localidade temporal é o fator predominante, isto é, a chances de acessar s informações do host da última URL processada é muito maior do que as do host de uma URL processada a 500 iterações atrás. A estrutura utilizada para o armazenamento é um hash map, uma estrutura de par chave-valor, que retorna o endereço resolvido e um objeto com as regras do robots, para uma dada URL, com complexidade de acesso $O(1)$, dada a natureza única da chave utilizada para recuperar o valor, a URL. Outra função importante deste componente é a função *boolean canResolveAddress(Url)*, que verifica se a URL passado como parâmetro, acessa algum diretório assinalado com a regra ***Disallow*** no robots.txt. Basicamente ele acessa o objeto do host na cache, e dentro deste objeto acessa uma lista que contém os diretórios assinalados com a regra ***Disallow***, e caso alguns dos diretórios do caminho desta URL esteja nesta lista a função retornará *false*, e o documento será ignorado.

3.3. Fetcher

O *fetcher* é um dos principais componentes da máquina. Traduzindo literalmente ao português: Buscador. Como o próprio nome diz, é o componente que realiza a busca das informações fazendo downloads em servidores web através do processamento de URL's chamadas "sementes". Após o componente do DNS e Robot Resolver essas URL's para os endereços IP's corretos, o fetcher então entra em ação para fazer a busca desses documentos a partir das "sementes"definidas.

3.3.1 Implementação do fetcher

A implementação do Fetcher, foi realizada no arquivo ***Fetcher.java***. A implementação é bem simples, pois facilitada pelo uso de uma biblioteca que facilita o download de arquivos HTML, conhecida como Jsoup. O fetcher possui duas funções principais e um construtor, a funcionalidade de cada uma delas é explicada abaixo:

- *Fetcher(URL)* Construtor público que cria uma nova instância do fetcher, que irá tratar a URL passada como parâmetro, neste construtor são realizadas algumas validações quanto ao formato da URL.
- *Document fecthDocument()* Esta função é responsável por realizar o download do documento HTML e armazená-lo em um objeto da biblioteca Jsoup, do tipo Document e retorná-lo.
- *void sendDocumentToParser()* Este método basicamente recebe os objetos do tipo Document retornados pela função anterior e os envia para o Parser. Ele espera pela resposta do Parser, que será uma lista de URL's e um corpo de texto do documento. As URL's são enviadas para a lista de URL's a serem coletadas e salvas no disco, os documentos salvos no disco.

Como podemos observar a partir da descrição existe para cada documento um Fetcher e um Parser tratando o download e recuperação das URL's do documento. O Fetcher quando instanciado tem sua execução já em paralelo associado a uma thread, e como ele cria a instância do Parser,

observamos que temos esse par correndo na mesma thread, o que facilita a troca de dados entre ambos os componentes.

3.4. Parser

Como o próprio nome diz, essa fase é a de "interpretação"dos documentos encontrados pelo fetcher. Pela definição da palavra *parser* é uma fase que irá examinar cuidadosa e detalhadamente os documentos, de forma a compreendê-lo e dar inicio a "transformação"dos dados encontrados em informação. Essas informações podem ser textos ou até novas URL's, as quais continuaram alimentando o coletor, que efetuará todo o processo novamente com essas novas URL's encontradas.

3.4.1 Implementação do fetcher

O conjunto de implementações deste componente também é muito simples, e também conta com recursos da biblioteca Jsoup. Consiste basicamente de uma função, a função *ParsedData parseDocument(Document)*. Esta função recebe um documento do tipo Document pertencente ao Jsoup, extrai as URL's e textos do documento e os retorna em um objeto ParsedData, objeto definido para facilitar o manejo dos dados parseados, que é um objeto que contém um arranjo de URL e Strings. Este arranjo e Strings na mais é do que o corpo do documento já pré-tokenizado, pois ele contém cada termo que ocorre no documento, ordenados na ordem em que ocorrem.

3.5. Filter

O componente de filtro é importantíssimo para o o processo de coleta. Nesse componente, o conteúdo encontrado na fase do *parser* será filtrado, de forma a decidir se uma URL extraída deve ser considerada para voltar ao processo de coleta, ou se o conteúdo do documento baixado deve ser guardado, pensando na ideia de que existem conteúdos que podem não ser considerados interessantes para o usuário, como web spams.

3.6. Scheduler

Este componente é responsável por definir e escalonar as URL's a serem processadas pelo coletor. Define-se uma politica de escalonamento que seja capaz de determinar prioridade entre as URL's, assim definindo qual URL deve ser processada primeiro. A cada documento obtido pelo coletor novas URL's são extraídas e enviadas a este componente que a cada inserção ou conjunto de inserções reorganiza sua fila de prioridade.

3.7. Outras implementações

Nesta seção abordaremos outras implementações que contém parte das funcionalidades do Scheduler e do Filter, que por serem ainda muito simples são abordadas de maneira conjunta.

A classe responsável pelo comportamento principal do Coletor está contida no arquivo *MainTest.java*. Neste arquivo temos o "kernel"do funcionamento principal do coletor. Temos uma estrutura de dados do tipo lista, que é usada para armazenar as URL's coletadas pelo coletor. Esta estrutura possui uma característica permitida pela Linguagem utilizada na implementação, o Java, ela é sincronizada, o que possibilita que múltiplas threads possam operar sobre ela sem a existência de conflitos, com a ideia da exclusão mútua. Nesta estrutura de dados a estratégia de escalonamento adotada é a FIFO.Esta estratégia foi escolhida por ser de fácil implementação, mas posteriormente outras estratégias mais eficazes serão consideradas. Inicialmente insere-se nesta lista um conjunto de URL's semente já predefinidas, através do método *initUrlListWithSeed()*. Foram elas :

- http://pt.wikipedia.org/wiki/Categoria:Pessoasvivas
- http://pt.wikipedia.org/wiki/Categoria:Instituições
- http://globo.com
- http://www.nytimes.com/
- http://www.dicionariodenomesproprios.com.br/
- http://www.yahoo.com
- http://dmoztools.net/

Logo após a inserção das sementes, inicia-se o processo de coleta, que ocorre na função principal do programa, que consiste de um loop infinito que vai lendo uma a uma as URLs da lista e criando instâncias de Fetcher paralelas em threads para processar esta URL. Para controle das threads criadas, foi utilizado um recurso do Java chamado ExecutorService, que nada mais é do que um pool de trabalho de threads, onde definimos uma quantidades de threads a serem criadas (no caso deste coletor foram 16 threads, constante esta definida de maneira empírica e dentro das condições de processamento que se possui) e a medida que processamento vai sendo enviado para esta pool, ela delega uma tarefa à alguma thread livre para realizar esta tarefa. A gerência da pool e realizada automaticamente, na medida que threads vão sendo liberadas e novas tarefas vão chegando, evitando assim problemas de desempenho. A lista de URLs vai sendo esvaziada na medida que as URLs vão sendo processadas, para que novas URL's sejam inseridas e não ocorra um overflow.O Fluxo de URL's também é considerado, pois se uma thread gera muito trabalho a inserção de URL's na lista é cessada até que um espaço considerável seja criado pelo consumo desta grande carga de trabalho.

4. Dados da coleta

4.1. Ambiente

A coleta foi realizada no seguinte ambiente:

- Intel® Core™ i5 3230M 2.60 GHz com turbo boost 3.20 GHz
- SSD Samsung Evo 120G
- Ubuntu 16
- 6Gb de memória RAM

4.2. Medidas da Coleta

A coleta ocorreu em um período de 6 horas 42 minutos e 19 segundos. Foram coletados 47.921 documentos, porém apenas 18.851 documentos foram armazenados em 93,7 Mb de dados no formato texto . As URL's coletadas formaram um conjunto de 182,6 Mb de dados no formato texto, contendo um total de 2495244 URL's coletadas.Cada documento levou em média 2034ms para ser coletado. A taxa média de coleta de documentos foi de de aproximadamente 30 documentos por minuto.

5. Implementações futuras

Algumas implementações futuras serão realizadas para otimizar e melhora o desempenho do coletor. São elas:

- Implementação de uma estratégia de escalonamento mais interessante
- Abordar estratégia de cache de endereços e robots mais complexas que envolvam armazenamento em memória secundária
- Implementação de um componente de filtragem que trate web spam.
- Abordar requisitos como freshness e o tratamento de entidades.
- Implementar um melhor tratamento de duplicatas.

6. Indexador

O indexador é o componente da máquina de busca responsável por construir estruturas de dados para armazenamento e acesso eficiente do conteúdo na base de dados, com o objetivo de aumentar o tempo de resposta das consultas. O indexador processa os documentos da base de dados, extraindo trechos do texto, como sentenças e expressões para posterior modificação. Esta modificação, trata de extrair os termos transformando estes elementos textuais em tokens. Logo após este tokens são tratados de maneira a tornar o processo de indexação e respostas a consultas mais eficientes, como por exemplo a remoção de trechos com pouco valor semântico. Como resultado deste processos duas novas estruturas de dados são produzidas: o **vocabulário**, que consiste no conjunto de todos os tokens armazenados, e um **índice de ocorrências**, que armazena para cada token contido no vocabulário os documentos da base de dados que contém ocorrências deste token. A estruturação do índice de ocorrências, é um fator determinante para eficiência tanto para o acesso das informações como para atender as consultas. Por isso a maneira como esta estrutura é criada e armazenada é fundamental para o êxito da máquina de busca, também deve ser levado em consideração nesto processo meios que possam auxiliar na identificação das entidades. Nos sub-tópicos a seguir serão dados mais detalhes a cerca disto e dos sub-componentes do indexador. Na Figura 2 observamos o diagrama da arquitetura dos componentes do indexador.

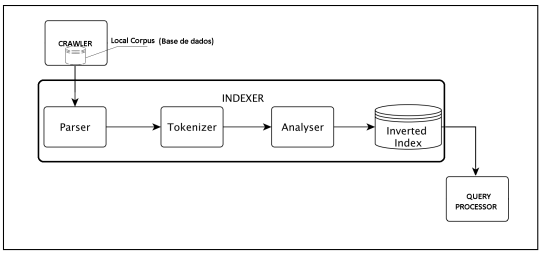

Figura 2. Diagrama dos Componentes da Arquitetura do Indexador da Máquina de Busca

6.1. Parser

Como na fase de coleta, o componente do *parser* tem a função de extrair informações de documentos locais, como sentenças, frases, que sofrerão transformações posteriores, especificamente no componente *tokenizer*.

6.2. Implementaç[a]o

6.2.1 Tokenizer

Este componente tem a função de, dada uma sequência de caracteres de dado documento, cortá-lo em pedaços, denominados *tokens*, jogando fora certos personagens, como a pontuação por exemplo. Um *token* pode ser compreendido como uma instância de uma seqüência de caracteres em algum documento particular que são agrupados como uma unidade semântica útil para processamento.

A implementação do *(*Tokenyzer), foi realizada no arquivo ***Tokenyzer.java***. De maneira simples, foram criadas duas funções e uma estrutura de dados principail, sendo a funcionalidade de cada uma delas explicada abaixo:

- *HashMap<String, Integer> occurrence* Estrutura de dados para mapear os strings com suas respectivas ocorrências no documento.

- *List<String> getTokens(File file)* Recebe um arquivo como paramêtro, o qual contém todas as palavras presentes no documento em questão. Este método realiza o "controle"do atual componente, basicamente lendo as linhas do arquivo, realizando um pequeno parse no mesmo de modo a extrair uma lista de *strings* que será passada para a função que *void getOcurrency(String[] lineTokens)*, a qual será descrita a seguir.
- *void getOcurrency(String[] lineTokens)* Esta função é responsável pelo processamento principal do componente. Iterando-se sob os strings contidos no array *lineTokens*, recebido como parametro, verifica-se se o *string* da iteração corrente ja está contido na estrutura de dados *occurrence*. Caso a verificação resulte em "falso", o elemento em questão é adicionado a estrutura, caso contrário ele tem seu valor de ocorrência incrementado na estrutura.

6.2.2 Analyzer

O componente *analyzer* tem o papel de efetuar algumas operações no texto em determinados tokens. Essas operações foram realizadas em duas funções principais:

- *String removeStopWords(String fileString)* Itera-se sobre as palavras contidas no arquivo removendo as *stopwords*. Foram criadas três listas contendo stopwords em português, inglês e espanhol respectivamente. AS palavras contidas nessas listas foram usadas para realizar a comparação e remoção.
- *String removeAccents(String fileString)* Mantendo uma lista de letras acentuadas, realiza-se uma varredura em todas as palavras do arquivo verificando se as mesmas contém algum acento, de modo a substituí-lo pela letra correta.

6.2.3 Vocabulary & Ocurrence Index

Basicamente são duas estruturas de dados. A primeira é usada para guardar os tokens selecionados. A Segunda guarda, para cada token do vocabulário, os documentos na coleção local que contém o token.Geralmente o índice de ocorrências é armazenado em um estrutura de índice invertido.

7. Processador de Consultas

No contexto de Recuperação de Informação processar uma consulta, significa responder a uma consulta requisitada pelos os usuários através de uma interface amigável. O usuário insere um conjunto de palavras e termos que carregam consigo a semântica da informação que se deseja encontrar e espera que como resultado seja retornado um conjunto relevante de respostas que satisfaçam a necessidade da consulta realizada. O componente responsável por isto é o processador de consultas. Após o usuário realizar a consulta a primeira operação a ser realizada é armazenar esta consulta em um *log de consultas* para posterior acesso. Após isto inicia-se o processo de compreensão da consulta, que consiste em realizar algumas operações para realçar a informação que o usuário precisa realizando operações como correções ortográficas, expansão de acrônimos,stemming etc. g(operações que serão abordas posteriormente) com o objetivo de reformular a consulta do usuário para obter melhores resultados. Após este processo ocorre o ranqueamento, que trata-se da operação de criar um modelo de recuperação para consulta e o conteúdo indexado a fim de entregar uma classificação de documentos relevantes, que satisfazem corretamente as necessidades de informação da consulta do usuário. Na Figura 3 está um diagrama que demonstra a arquitetura e o funcionamento deste componente. Mais detalhes sobre os sub-componentes e o funcionamento do processador de consultas são abordados a seguir.

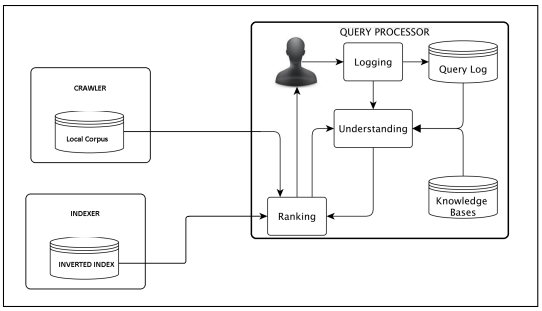

Figura 3. Diagrama dos Componentes da Arquitetura do Coletor da Máquina de Busca

7.1. Interface

Este componente consiste basicamente em uma interface gráfica provida ao usuário para que ele interaja com a máquina de busca. Geralmente a interface inicial com o usuário contém um campo de texto para que ele insira os termos da busca e um botão que acione a busca. Pode conter também alguns campos opcionais de filtros que ajudem a guiar a busca. Outra interface é a pela qual os resultados da busca são apresentados, que nada mais é do que uma lista ordenada por relevância de documentos com algumas meta-informações e uma pequena descrição ou trecho que faça referência ao conteúdo. Algumas interfaces mais sofisticadas podem apresentar imagens sobre a busca ou caixas de conteúdo.

7.2. Logger

Componente responsável por registrar as buscas realizadas pelo usuário. As consultas são armazenadas pelo fato de elas contribuírem significativamente para o processo de

$$score(q,d) = \sum_{i=1}^{|q|} idf(q_i) \cdot \frac{tf(q_i,d) \cdot (k_1+1)}{tf(q_i,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl}}$$

Figura 4. Fórmula do cálculo de score do BM25

compreensão da consulta, ou na determinação de relevância de determinados conteúdos, podendo auxiliar também na construção do ranking. Este componente também é capaz de armazenar alguns resultados pré-carregados para consultas muito frequentes ou de grande relevância.

7.3. Expansion(Understanding)

Neste componente é realizado um processo de modificação dos termos da consulta do usuário, para aumentar a eficácia e alcance das consultas como objetivo de obter resultados de maior valor para o usuário. Várias técnicas podem ser abordadas nesta fase, sendo as principais citadas e explicadas abaixo:

- Stemming: Esta técnica busca encontrar radicais comuns em um grupo de termos é associar este grupos de termos armazenando esta associação, assim substituindo estes termos pelo radical na consulta.Por exemplo diminutivos e aumentativos sofrem a associação com o radical da palavra na sua forma normal.

- Speeling Correction: Processo de correção de erros de gramática contidos nos termos inseridos pelo usuário na consulta.

- Query Reduction: Processo de remoção de termos com baixo valor semântico. Termos que agregam pouco significado com preposições, conectivos etc.

7.4. Ranking

Este componente é responsável por gerar a lista de documentos ordenado por relevância ao usuário.Uma série de métodos podem ser utilizados para determinar a relevância de um documento para uma determinada busca. Elas podem abordas de um ponto de vista geral para um grupo de usuários ou até mesmo individual baseado no perfil de um usuário.

7.4.1 Implementação do Ranking

O modelo escolhido para implementação do ranking foi o BM25(Best Match 25)[**?**]. O BM25 é um modelo que combina características de diversos modelos de ranking da literatura.Na Figura 4 temos a fórmula para o cálculo do score do modelo BM25.

7.4.2 A máquina de busca

A máquina teve sua lógica implementada 100% em Java. A interface do usuário foi implementada em versão web, utilizando basicamente HTML com Bootstrap Framework e PHP. Visando uma usabilidade maior, foi implementada de maneira minimalista e 100%responsiva, isto é, ajustável aos mais váriados tamanhos de tela: tablets, smartphones, monitores e etc. A comunicação entre a interface e o backend foi realizada através da criação de uma API Restful. Esta API foi criada utilizando o framework Java Jersey, e esse sistema foi submetido à amazon aws, de modo que a comunicação é feita através de endpoints disponibilizados pela mesma. A máquina de busca, de nome *GunMar*, está disponível no endereço http://www.ibckoinonia.com.br/gunmar/.

8. Síntese da arquitetura

Como descrito, a arquitetura é baseada em três componentes principais: coletor, indexador e o processador de consultas. Basicamente a fase de coleta, como o próprio nome diz, faz a busca e download dos documentos que são tratados e entregues ao indexador, que já trabalha com documentos mais "legíveis". Essa legibilidade facilita no processo de extração das informações e tratamento para serem mostradas ao usuário de forma amigável no processador de consultas.

9. Referências

BRANDÃO, Wladmir Cadoso. Exploiting entities for query expansion 2013 (Tese de Doutorado).

Olston, Christopher; Najork, Marc. Web Crawling. Foundations and Trends in Information Retrieval, 4(3):175-246, 2010.

Jsoup,Open source Java HTML parser, with DOM, CSS, and jquery-like methods for easy data extraction;http://www.jsoup.org;