Web crawler em Java

coleta e filtragem de conteúdo da Web

Conheça o conceito de web crawler e aprenda como criar um agente especializado utilizando tecnologias Java, HttpComponents e JAX-RS

Dado o volume atual de dados existentes na Web, um processo automático e metódico de indexação de conteúdo se faz extremamente necessário. E é nesse cenário que o Web crawler se faz presente buscando, filtrando e muitas vezes persistindo conteúdo. Um dos mais poderosos web crawlers conhecidos é o googlebot, o crawler web da Google, ele é responsável por vasculhar a web, procurando por novas páginas para indexar e analisar se as páginas já existentes foram devidamente atualizadas.

Um Web crawler, que também é conhecido como web spider ou web robot, é um agente de software utilizado para automatizar pesquisa, captura e extração de dados. Trafegados geralmente em protocolos HTTP junto com o modelo TCP/IP, a maioria dos robots desenvolvidos são voltados para a web tradicional onde documentos html conectados por hyperlinks, tags e palavras chave são usados para rastrear e atender necessidades específicas como: download de imagens e arquivos, validação automática de códigos, mineração de endereços de e-mail, coleta de conteúdos específicos para cotação de preços de um determinado produto ou serviço e também pode ser desenvolvido para agregar dados de diversas fontes da internet podendo persistir essas informações em um banco de dados.

Esses agentes não estão limitados a uma linguagem específica, na verdade, desde que seja possível extrair e aplicar expressões regulares em cima dos conteúdos obtidos, qualquer linguagem pode ser utilizada para a implementação de web crawlers. No entanto, agora, não estamos levando em consideração a viabilidade de se implementar um crawler em cada linguagem. Cada uma possui suas particularidades, o que pode dificultar ou facilitar o desenvolvimento. Dito isso, para este artigo o Java foi a linguagem escolhida apenas por familiaridade e conveniência. O que não será um problema, já que os princípios gerais para qualquer linguagem estarão presentes.

O problema da informação

Para entendermos o porquê da prática do crawling, precisamos falar um pouco sobre o desenvolvimento da internet. Não iremos abordar a questão de forma técnica, pois o que é importante não é tanto o como, mas sim o quanto.

De acordo com a consultoria Jess3, o número de usuários da rede em 1995 já era pouca coisa maior que 45 milhões de pessoas e, de acordo com a mesma consultoria, o número em setembro de 2009 já era de 1,73 bilhões de pessoas. De qualquer ponto de vista esse é um número impressionante, mas que não diz muita coisa sozinho. No entanto, se juntarmos a ele o número de informação disponível através da web teremos um problema real.

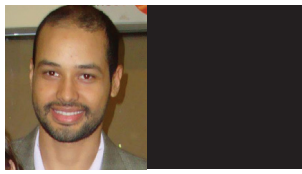
Uma pesquisa realizada pela University of Southern California intitulada “The World’s Technological Capacity to Store, Communicate, and Compute Information”, chegou a conclusão que, em 2002, a

Rodrigo Ribeiro | rdccr05@gmail.com

Atua há nove anos com análise e desenvolvimento de sistemas. Bacharel em Sistemas de Informação e Especialista em Desenvolvimento de Sistemas Web pelo IESAM. Possui certificações internacionais SCJP e OCWCD. Atualmente trabalha como consultor Java e PHP na Discover Technology.

Luderson Costa | luderson@gmail.com

Atua há um ano como desenvolvedor Objective-C e PHP. Cursou até o terceiro ano de Ciências da Computação pela UNIC, MBA em mercado de capitais. Atualmente trabalha como consultor PHP/Magento na Discover Technology.



Neste artigo abordaremos um pouco sobre a problemática que nos leva a pensar em indexadores automáticos, os principais conceitos sobre o que é um Web crawler, as tecnologias utilizadas como exemplo e sua aplicação prática, num elemento especializado que fará a extração de algumas informações livres de usuários do github.com, tais como: lista de repositórios, foto do usuário, urls para clonagem de projetos e download dos arquivos do repositório indexado.

quantidade de informação armazenada digitalmente superou em quantidade toda a informação armazenada de forma analógica. E como a informação analógica, a informação armazenada de forma digital precisa ser indexada para facilitar seu consumo, e é aí que entra o web crawling.

O Crawling

Dada a quantidade de informações disponíveis na Web, é impossível fazer uma indexação de forma manual, como se faz frequentemente em bibliotecas, portanto, esse processo deve ser feito de forma automatizada e sistemática. O agente responsável por essa tarefa é normalmente chamado de Crawler, daí vem a denominação da ação que esse agente realiza, o crawling.

O crawling não está, necessariamente, ligado à Web, no entanto é nela que se faz o maior uso dele, devido aos motivos já expostos.

A estrutura básica de funcionamento de um crawler não é muito complexa, apesar do mesmo não poder ser dito sobre sua implementação. Veja um modelo da estrutura básica de um crawler na figura 1.

As etapas giram em torno de uma requisição de conteúdo, extração da informação necessária, processo de indexação e persistência. A etapa de requisição e extração é inerente a qualquer processo de crawling, independentemente do objetivo e tecnologia utilizada. Já a indexação e persistência podem ou não ocorrer, e quando ocorrem normalmente são aplicados de forma muito específica, dependendo dos objetivos.

Requisições

Como dissemos anteriormente o crawling não está amarrado ao conteúdo web, ele pode atuar sobre diversos tipos de fontes. Desse modo ele não pode estar restrito por um protocolo específico, como o

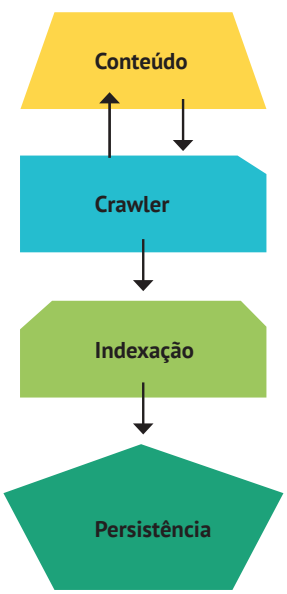


Figura 1 . *Fluxo de um Crawler.*

HTTP. É verdade, então, que se pode trabalhar em cima de SMTP, FTP, entre outros.

Através do protocolo escolhido o agente faz, então, uma requisição à fonte de dados, que acaba por retornar o conteúdo desejado, que será trabalhado para a extração das informações. O protocolo utilizado neste artigo será explicado de forma mais técnica mais adiante.

Extração através de Expressões Regulares

Há várias formas de se extrair dados de um documento. No entanto, ao lidar com conteúdo dinâmico é provável que aplicar expressões regulares seja a melhor forma.

Expressões regulares são padrões que lhe permitem localizar de forma flexível alguma sequência de caracteres. Isso significa que, com um mesmo padrão, é possível você localizar uma sequência de caracteres específica em qualquer lugar de um documento.

Por exemplo, é possível localizar qualquer ocorrência da sequência "car" dentro de um documento independentemente de fazer parte das palavras "carro", "carreta", "corretor" ou mesmo sozinha "car". Desse modo é possível resgatar qualquer dado que seja necessário e processá-lo conforme suas necessidades. Outro exemplo, mais prático, é a validação de data, vejamos abaixo:

Listagem 1. Expressão regular usada para validação de data.

```
^([1-9]|0[1-9]|[1,2][0-9]|3[0,1])/([1-9]|1[0,1,2])/\d{4}$
```

Esse padrão valida qualquer data no formato "99/99/9999", desse modo é possível confirmar se algum campo de data foi digitado corretamente, ou se existe alguma data no conteúdo de um documento.

A correta aplicação das expressões regulares será determinante na eficiência de um Crawler, tanto em sua abrangência quanto no retorno do conteúdo procurado.

Questões Legais

As questões de legalidade da prática do web crawling são, ainda, obscuras. E aqui deve ficar claro o que foi dito, isso vale principalmente para o conteúdo proveniente da Web. Já que pode existir um crawler varrendo a intranet de uma empresa, o que mantém os assuntos referentes ao acesso ao conteúdo, nesse caso, restrito a uma esfera não pública.

Nas questões do conteúdo proveniente da internet, a legislação é própria de cada país, o que aumenta muito as variantes. De qualquer forma é possível dividir essas questões em éticas e de lei positivada.

Nessa divisão é possível encontrar, pelo menos, quatro questões importantes: serviço, custo, privacidade e copyright.

Serviço e custos estão mais ligados à ética. Crawlers podem afetar a qualidade de repostas dos servidores ao fazerem várias requisições em um curto espaço de tempo. Isso afeta, ou pode afetar, a qualidade do serviço prestado aos consumidores finais, os usuários, deixando-os mais lentos. Eles podem aumentar também o consumo de banda, o que pode gerar custos extras aos proprietários de páginas.

A privacidade é talvez a questão mais difícil de se definir. Normalmente o que está disponível na internet é público. Qual a diferença entre acessar uma página através do navegador ou através de um programa que varre seu conteúdo? Aqui, talvez, o maior problema é o que se faz depois com a informação obtida. A prática de spam talvez exemplifique bem esse problema. E-mails podem ser buscados através de crawlers específicos para a função e depois usados em listas de malas diretas.

Já o copyright é o maior suporte legal à regulamentação do web crawling. Uma prática comum aos crawlers é a cópia e armazenamento de conteúdo com direitos reservados sem a permissão do proprietário.

Para aqueles que querem evitar o máximo de problemas legais e morais, uma recomendação é seguir as determinações do arquivo robots.txt. Esse protocolo usado largamente desde 1994 impõe algumas regras para o clawling so seu domínio, determinando o que pode ou não ser acessado. Por exemplo:

```
User-agent *
Disallow /teste/
```

Nesse caso não será permitido fazer a leitura do conteúdo do diretório "teste", e qualquer um de seus subdiretórios.

No entanto é bom lembrar que a leitura e obediência aos parâmetros desse arquivo não é obrigatória, é apenas uma boa prática para aqueles que desenvolvem crawlers.

Programando um Web Crawler

O HTTP (Hyper-Text Transfer Protocol) é talvez o protocolo mais importante usado na World Wide Web. Navegadores de internet, Web Services e o crescimento da computação em rede continua a expandir o papel do HTTP, e com isso o aumento do número de aplicações que necessitem de suporte a este protocolo. A web é um campo bastante interessante para pesquisas de mineração de dados, devido a grande quantidade de informação disponível na internet.

Uma técnica bastante conhecida é o Data Mining, que consiste no processo de descobrir informações relevantes, como padrões, associações, mudanças e estruturas, em grandes quantidades de dados armazenados em repositórios de informação. Fazendo uma conversão para o contexto do artigo, a técnica utilizada seria o Web Mining, que pode se entender

como uma extensão de Data Mining aplicado a dados da internet.

O que é Web Mining?

Web Mining envolve técnicas de recuperação de Informação, estatística, inteligência artificial e mineração de dados. É uma maneira usada para descobrir e extrair automaticamente dados de serviços e documentos disponíveis na internet, em geral, as principais tarefas de web mining são:

- Busca de Documentos: a técnica usada nessa tarefa é a de Recuperação de Informação que consiste no processo de extrair dados a partir de sites web se valendo do conteúdo de documentos HTML, navegando entre palavras chaves e tags eliminando-as e recuperando os textos contidos nessas estruturas.
- Seleção e pré-processamento: o pré-processamento envolve qualquer tipo de transformação da informação obtida na busca, como, por exemplo, corte de textos.
- Generalização: com a utilização de técnicas de inteligência artificial e mineração de dados, possui a tarefa de descobrir de forma automática padrões de um ou entre vários sites web.
- Análise: a análise é aplicada geralmente em todas as tarefas que consiste em validar e interpretar os padrões minerados.

Protocolo HTTP + JAVA

Originalmente, o Java vem com bibliotecas para trabalhar e manipular dados trafegados no protocolo http. Seguindo as especificações w3c para protocolos HTTP é possível você criar seu próprio servidor sem a necessidade de APIs externas. Geralmente para esse tipo de tarefa utiliza-se as classes URL, HTTPUrlConnection e Socket do pacote java.net. Pensando pelo lado acadêmico é interessante aprender como se criar um servidor http, criando suas próprias classes HttpClient, conectando via sockets etc. Mas quando o tema produtividade é citado, buscar alternativas maduras no mercado é a melhor opção. Atualmente existem frameworks robustos que facilitam a interação e manipulação de dados trafegados na web.

HttpComponents da Apache

O HttpComponents da Apache proporciona um robusto suporte para protocolos HTTP, possui componentes com poder para construir aplicativos cliente e servidor, como navegadores web, web crawlers, proxies HTTP, comunicação distribuída etc. Entre os componentes destacam-se o HttpCore, que atua no baixo nível de transporte HTTP e que pode ser usado para construir cliente personalizado, serviços do lado do servidor e suporte a modelo de I/O. HttpClient, é um agente com base no httpcore, compatível com aplicações HTTP fornece componentes reutilizáveis para a autenticação do cliente, manter estado e gerenciamento de conexões, consumo de conteúdo html etc.

O Web Crawler

Nesta fase do artigo, iremos desenrolar o desenvolvimento do web crawler, a tarefa de web mining do nosso exemplo será do tipo busca de documentos e será utilizado o componente HttpClient da Apache com o recurso para consumo de páginas html, Expressões Regulares para a recuperação específica da informação e criaremos um websevice com JAX-RS para disponibilizar os dados recuperados pelo agente de software. Serão extraídas algumas informações livres de usuários do github.com. O agente será responsável por capturar o avatar, lista de repositórios, os links para comandos git clone de projetos e a url para download dos arquivos ZIP do repositório indexado.

Mão na massa: Fase 1

A primeira fase para a criação de um *web crawler* se dá pela avaliação do conteúdo a ser extraído pelo agente. Nosso primeiro passo então é analisar a página principal do GitHub para extrair somente os links do menu de navegação. Vamos avaliar a seguinte url: https://github.com/.

Listagem 2. Expressão regular usada para extrair conteúdo da home do github.

```
<[^>]*?class=\"(?:explore|search|features|blog)
[^>]*?\"><a[^>]href=\"(.*?)>(.*?)</a></[^>]*?>
```

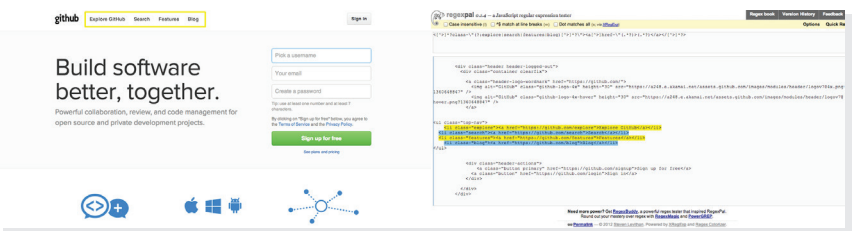


Figura 2. *Página principal do GitHub com expressão regular de extração sendo executada no site http://regexpal.com/.*

Agora o próximo passo é criar a regex para extrair conteúdo específico da página de usuários do github. Vamos avaliar a próxima url: https://github.com/rdccr e extrair o avatar e o link para os repositórios. Veja figura 3.

Listagem 3. Expressões regulares usadas para extrair o avatar e a url de repositórios do usuário github.

```
AVATARED: <div\\sclass=\"avatared\">[^>]*?<a\\
shref=\"(.*?)\"
REPOSITORIES_TAB: [^<a\\shref=\"].*?\\?tab=repositories
```

Nosso último passo é criar a expressão para extrair a lista de repositórios de https://github.com/rdccr/repositories e seus respectivos conteúdos. Como na figura 4.

Listagem 4. Expressões regulares usadas para extrair a lista de repositórios e conteúdo de cada repositório.

```
REPOSITORIES_LIST: <h3>[^>]*?<span[^>]*?>.*?</
span>[^>]*?<a\\shref=\"(.*?)\">(.*?)</a>[^>]*?</h3>
DOWNLOAD_ZIP: <a\\shref=\"(.*?)\"[^>]*?><span[^>]*?></
span>ZIP</a>
CLONE_URL: <[^>]*?\\sclass=\"(?:http_clone_url|public_
clone_url|private_clone_url)[^>]*?\">[^>]*?<a\\
shref=\"(.*?)\"[^>]*?>(.*?)</a></[^>]*?>
```

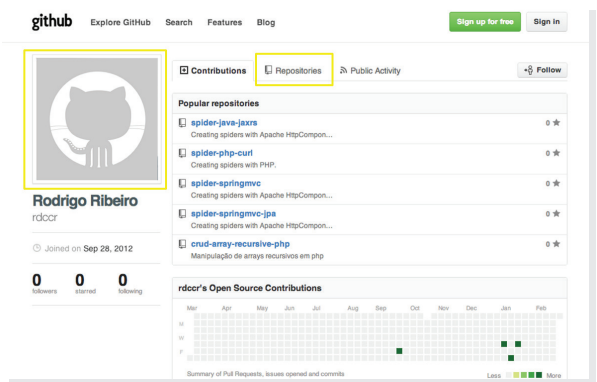


Figura 3. *Página carregada pela url https://github.com/rdccr*

Mão na massa: Fase 2

Agora com as expressões regulares devidamente criadas vamos começar a desenvolver programaticamente o web crawler. Na fase 2 iremos criar os beans que irão mapear os resultados capturados e o motor de busca do crawler. Seguem abaixo os fragmentos de código com a estrutura dos beans.

Listagem 5. Fragmentos de código com a estrutura dos beans.

```
//Bean responsavel por mapear a home do github.
public final class GitHubHome implements Serializable {
    private static final long serialVersionUID = 1L;
    private String title;
    private String url;
// Getters e Setters
}


//Bean responsavel por mapear repositorio.
public final class GitHubRepository implements
Serializable {
    private static final long serialVersionUID = 1L;
    private String title;
    private String url;
    private String zip;
// Getters e Setters
}

//Bean responsavel por mapear links para comandos git
clone.
public final class GitHubCloneUrl implements
Serializable {
    private static final long serialVersionUID = 1L;
    private String title;
    private String url;
// Getters e Setters
}
//Objeto responsavel por mapear resultado do github.
public final class GitHub implements Serializable {
    private static final long serialVersionUID = 1L;
```

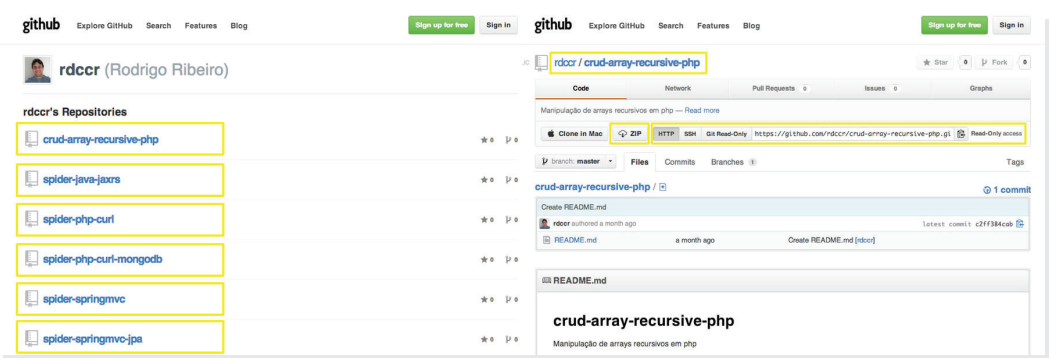


Figura 4. *Página da lista de repositórios https://github.com/rdccr/repositories e a página do repositório https://github.com/rdccr/crud-array-recursive-php usado para ilustrar o conteúdo a ser capturado.*

```
    private List<GitHubRepository> repositories;
    private List<GitHubCloneUrl> cloneUrls;
    private String urlAvatared;
    private String urlTabRepositories;
// Getters e Setters
}
/*Constantes regex para parseamento do github. */
public final class GitHubPattern {
// Expressao usada para recuperar pagina de repositorios.
    public static final String REPOSITORIES_TAB =
        "[^<a\\shref=\"].*?\\?tab=repositories";
    // Expressao usada para recuperar foto do usuario.
    public static final String AVATARED =
        "<div\\sclass=\"avatared\">[^>]*?<a\\shref=\"(.*?)\"";
    // Expressao usada para recuperar lista de repositorios.
    public static final String REPOSITORIES_LIST =
        "<h3>[^>]*?<span[^>]*?>.*?</span>[^>]*?
         <a\\shref=\"(.*?)\">(.*?)</a>[^>]*?</h3>";
    // Expressao usada para recuperar arquivo zip para
download.
    public static final String DOWNLOAD_ZIP =
        "<a\\shref=\"(.*?)\"[^>]*?><span[^>]*?></span>
        ZIP</a>";
    // Expressao usada para recuperar links para o
comando git clone.
    public static final String CLONE_URL =
        "<[^>]*?\\sclass=\"(?:http_clone_url|public_clone_
        url|private_clone_url)[^>]*?\">[^>]*?<a\\
        shref=\"(.*?)\"[^>]*?>(.*?)</a></[^>]*?>";
    // Expressao usada para recuperar menu principal da
home do github.
    public static final String GITHUB_HOME =
        "<[^>]*?class=\"(?:explore|search|features|blog)
        [^>]*?\"><a[^>]href=\"(.*?)>(.*?)</a></[^>]*?>";
    // Expressao usada para recuperar erro caso usuario
submetido nao exista

}
```

O próximo fragmento de código será referente ao da interface IGitHubSearchEngine que foi criada para conter o contrato com as implementações das assinaturas search(), findUrlAvatared(), findUrlTabRepositories(), downloadZipFile(), findCloneUrls(), findMenuGitHubHome() que o motor de busca do crawler deverá seguir para seu funcionamento. Segue abaixo:

Listagem 6. Contratos que devem ser seguidos pelo Crawler.

```
public interface IGitHubSearchEngine {
    // Constant com o dominio github.
    final String DOMAIN = "https://github.com";

    // Clausula para executar de pesquisa padrao.
    GitHub search(HttpClient httpclient, HttpGet httpget,
        ResponseHandler<String> responseHandler);

    // Clausula para buscar foto de usuario github.
    String findUrlAvatared(HttpClient httpclient, HttpGet
      httpget, ResponseHandler<String>
        responseHandler);

    // Clausula para buscar url parent da busca de
repositorios
    String findUrlTabRepositories(HttpClient httpclient,
      HttpGet httpget, ResponseHandler<String>
        responseHandler);

    // Clausula para buscar url de arquivo ZIP de
repositorio.
    String downloadZipFile(HttpClient httpclient, HttpGet
      httpget, ResponseHandler<String>
        responseHandler);

    // Clausula para buscar links para comandos git clone.
    List<GitHubCloneUrl> findCloneUrls(HttpClient
      httpclient, HttpGet httpget,ResponseHandler
        <String> responseHandler);

    // Clausula para buscar links da pagina principal do
github.
    List<GitHubHome> findMenuGitHubHome(HttpClient
      httpclient, HttpGet httpget,
        ResponseHandler<String> responseHandler);
}
```

A interface descrita acima representa a estrutura do agente e os métodos que serão implementados possuem os seguintes argumentos: HttpClient responsável por trafegar no protocolo http, HttpGet responsável por consumir via get a url que será utilizada pelo cliente http e o ResponseHandler<String> responsável pela resposta do conteúdo html acessado. A manipulação desses objetos é a base para o funcionamento do web crawler desenvolvido.

Listagem 7. Implementação do método findMenuGitHubHome().

```
// WebSpider GitHubSearchEngine.
public final class GitHubSearchEngine implements
IGitHubSearchEngine {

    // Referencias para motor de busca
    private ICompileRegex regex;
    private Matcher matcher;
```

```
    //Variaveis de instancia
    private String urlTabRepositories;
    private String urlAvatared;
    private String urlDownloadZipFile;
    private List<GitHubRepository> listGitHubRepository;
    private List<GitHubCloneUrl> listGitHubCloneUrl;
    private final GitHub gitHub;
    private List<GitHubHome> listGitHubHome;

    // Construtor com argumento
    public GitHubSearchEngine(ICompileRegex regex) {
        this.regex = regex;
        this.gitHub = new GitHub();
    }
    // outros métodos...

    /**
     * Metodo utilizado para recuperar lista de links da
home do github.
     */
    public List<GitHubHome> findMenuGitHubHome(
        HttpClient httpclient, HttpGet httpget,
        ResponseHandler<String> responseHandler) {
        try {
            // Cria referencia para lista de objetos
GitHubHome.
            this.listGitHubHome =
                new ArrayList<GitHubHome>();
            // Executa regex na resposta do httpclient.
            this.matcher = regex.execute(
                GitHubPattern.GITHUB_HOME, httpclient.
                execute(httpget, responseHandler));
        /*
         * Percorre casamento do padrao, cria uma
         * referencia de GitHubHome,
         * populando e adicionando lista listGitHubCloneUrl
         */
            while (this.matcher.find()) {
                // Cria referencia do Bean de menu da home.
                GitHubHome gitHubHome =
                    new GitHubHome();
                // Seta Valores para a referencia do bean.
                gitHubHome.setUrl(this.matcher.group(1));
                gitHubHome.setTitle(this.matcher.group(2));
                // Adiciona a referencia atual a lista de itens
                // de menu.
                this.listGitHubHome.add(gitHubHome);
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // Fecha a conexao de httpclient.
            httpclient.getConnectionManager().shutdown();
        }
        return this.listGitHubHome;
    }
    // Outros médotos...
}
```

Analisando o código da Listagem 3, podemos verificar que a lógica de busca proposta funciona da seguinte forma. Em uma classe criada para encapsular as tarefas de compilação da expressão regular, é passado como parâmetro o padrão a ser usado referenciado pela constante GitHubPattern.GITHUB_HOME e a resposta da execução do client http representado por httpclient.execute(httpget, responseHandler). O resultado desse casamento de dados será passado para um objeto matcher e, caso this.matcher.find() seja verdadeiro, significa que a expressão regular obteve retorno, serão adicionadas novas informações ao objeto GitHubHome. Note que no finally fechamos a conexão do HttpClient.

Mão na massa: Fase 3

Nesta última fase vamos criar um webservice para disponibilizar acesso às ações do web crawler, podendo assim as informações serem consumidas e, caso seja necessário, persistidas em algum modelo de dados. Para a criação do webservice rest será utilizado o JAX-RS com Jersey para gerar um retorno no formato json. Segue trecho do código:

Listagem 8. Web service para disponibilizar as informações.

```
@Path("/github")
public final class GitHubService {

    /**
     * Metodo do servico que retorna links da pagina
principal do github.com
     * ACESSAR DA SEGUINTE FORMA: http://
localhost:8080/crawler/service/github/
     */
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Map<String, List<GitHubHome>>
    findMenuHome() {

        // Dominio do github.
        String gitHubHomeDomain =
            IGitHubSearchEngine.DOMAIN;

        // Cria referencia para map de objetos GitHubHome.
        Map<String, List<GitHubHome>> menuHome =
            new HashMap<String, List<GitHubHome>>();

        // Cria referencia para GitHubSearchEngine.
        IGitHubSearchEngine gitHub =
            new GitHubSearchEngine(new CompileRegex());

                    /*
```

```
                * Cria referencia listGitHubHome para
                  lista de objetos GitHubHome com o
                * retorno de
                  gitHub.findMenuGitHubHome().
                */
        List<GitHubHome> listGitHubHome =
            gitHub.findMenuGitHubHome(
                new DefaultHttpClient(),
                new HttpGet(gitHubHomeDomain),
                new BasicResponseHandler());

        // Adiciona valor ao map.
        menuHome.put(gitHubHomeDomain,
            listGitHubHome);

        return menuHome;
    }
}
```

Para acessar os serviços localmente digite no seu navegador: http://localhost:8080/Spider/service/github/para extrair os itens do menu principal da home do site github, ou http://localhost:8080/Spider/service/github/seu-usuariopara extrair sua foto, lista de repositories e arquivos para download. Veja figuras 5 e 6.

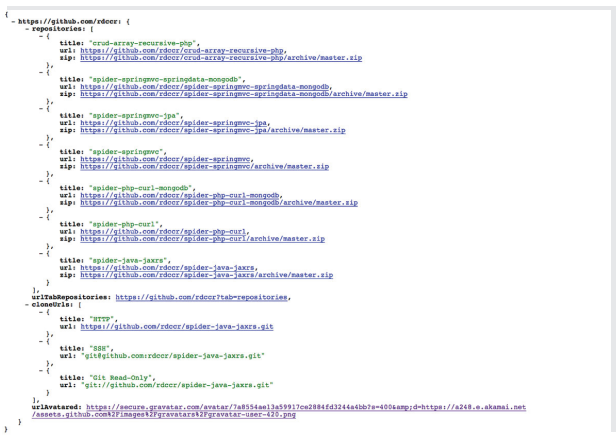


Figura 5. *Resposta do webservice rest acessado pela url http://localhost:8080/Spider/service/github/seu-usuario exibindo o json com o conteúdo extraído.*

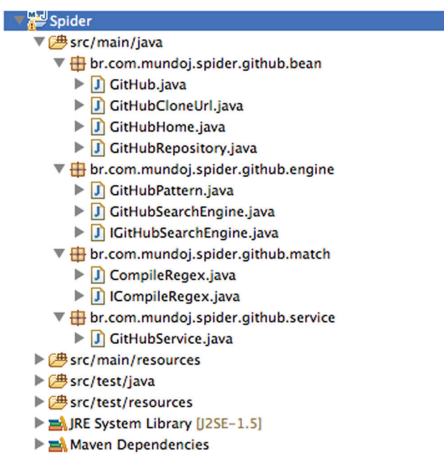


Figura 6. *Estrutura de pacotes do web crawler.*

Considerações finais

O propósito deste artigo foi mostrar o porquê da prática do web crawling e sua importância no contexto da web atual, levando em consideração seu tamanho e necessidade do consumo de seu conteúdo.

Vimos também os conceitos básicos de qualquer crawler, independentemente de linguagem e tecnologia. Aproveitamos e exemplificamos de forma prática a implementação em Java de um agente específico, que faz a captura e filtragem de dados, livres, do github.com.

Esperamos que assim o leitor tenha a base para iniciar uma implementação de um crawler em qualquer linguagem.

/para saber mais

> *Na edição número 24 da revista MundoJ é abordada a utilização de Expressões Regulares com java.*

/referências

> *"Mastering Regular Expressions" (Jeffrey Friedl).*

> *Livro - Expressões Regulares: Uma Abordagem Divertida (Aurelio Marinho Jargas)*

> *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Bing Liu)*

> *http://www.scribd.com/doc/52799786/Definicao-Web-Mining*

> *http://www.dct.ufms.br/~mzanusso/Data_Mining.htm*

> *http://hc.apache.org/*

> *http://regexpal.com/*

> *http://news.usc.edu/#!/article/29360/How-Much-Information-Is-There-in-the-World*

> *http://www.sciencemag.org/content/332/6025/60*

> *http://jess3.com*

> *http://en.wikipedia.org/wiki/Web_scraping#Legal_issues*

> *Web Crawling Ethics Revisited: Cost, Privacy and Denial of Service1,*

> *http://carl.cs.indiana.edu/fil/WebSec/mining_social_networks.html*