Universitatea Tehnica a Moldovei
Facultatea Calculatoare Informatică și Microelectronică

Departamentul ISA

Raport

Lucrarea de laborator Nr. 3
la Programarea în rețea

Tema:Protocolul HTTP. Proiectare și programare aplicație client

A elaborat st. gr. TI-144
D. Gorduz
A verificat lect. sup.
S. Ostapenco

Chișinău 2017

Scopul lucrării

Înțelegerea protocolului și aplicarea acestuia. Sarcinile variază între client simplu HTTP la aplicații-utilitare de colectare a informației în Web.

Obiectiv

Înțelegerea protocolului HTTP și rolul acestuia în comunicarea în Web, studiul componentelor Java utile în aplicarea protocolului HTTP; obiectivul specific lucrării constând în crearea unei aplicații client Java care ar interacționa cu server Web prin intermediul metodelor HTTP studiate.

Link la repozitoriu

https://github.com/GorduzDaniel/Lab3_PR

Noțiuni teoretice

În corespundere cu modelul de referință ISO/OSI HTTP este un protocol de nivel aplicație și definește organizarea fundamentală a ceea ce se numește World Wide Web (WWW). Protocolul, fiind proiectat să permită diverse elemente intermediare de comunicare dintre aplicațiile Web, a indus dezvoltarea unui sistem informațional hypermedia, distribuit și colaborativ. Transferul informației indiferent de tipul datelor, dar și deschiderea spre alte tehnologii a favorizat utilizarea pe larg a acestui protocol. [1]

Metodele HTTP definesc operațiile de efectuat la server asupra resursei la primirea cererii. Acestea fiind gândite într-un context mai larg decât „cererea și primirea unei pagini Web” au permis în timp interacțiuni mai complexe, ce stau la baza unor sisteme Web distribuite. Numele metodei este indicată în prima linie de interogarea și este dependentă de registru.

Metodele disponibile sunt: [2]

1. **GET**: este cea mai folosită metodă, fiind utilizată atunci când serverului i se cere o resursă.
2. **HEAD**: se comportă exact ca metoda GET, dar serverul returnează doar antetul resursei, ceea ce permite clientului să inspecteze antetul resursei, fără a fi nevoit să obțină și corpul resursei.
3. **PUT**: metoda este folosită pentru a depune documente pe server, fiind inversul metodei GET.
4. **POST**: a fost proiectată pentru a trimite date de intrare către server.
5. **DELETE**: este opusul metodei PUT.
6. **TRACE**: este o metodă folosită de obicei pentru diagnosticare, putând da mai multe informații despre traseul urmat de legătura HTTP, fiecare server proxy adăugându-și semnătura în antetul Via.

7. **OPTIONS**: este folosită pentru identificarea capacităților serverului Web, înainte de a face o cerere.
8. **CONNECT**: este o metodă folosită în general de serverele intermediare.

Mersul lucrării

Pentru efectuarea acestei lucrări de laborat a fost utilizată pachetul de instrumente de la Apache: HttpClient.[3]

Pentru început, în urma analizei din browser a unui GET, putem să reconstruim antentul browserului, ca să fie simulat comporatamentul unui browser. Acest lucru se poate observa în figura 1.

```
public String sendGet(String url) throws Exception {

    HttpGet get = new HttpGet(url);
    get.setHeader( name: "Accept",
            value: "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8");
    get.setHeader( name: "Accept-language",  value: "en-US,en;q=0.8");
    get.setHeader( name: "Cache-control",  value: "max-age=0");
    get.setHeader( name: "User-Agent",  value: "Mozilla/5.0");

    HttpResponse response = client.execute(get);

    System.out.println("Send GET to " + url);
    System.out.println("Response code " + response.getStatusLine().getStatusCode());
    BufferedReader br = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = br.readLine()) != null ){
        result.append(line + "\n");
    }
    if(cookies != null && response.getFirstHeader( s: "Set-Cookie") != null){
        cookies = response.getFirstHeader( s: "Set-Cookie").toString();
    }
    Thread.sleep( millis: 1000);
    return result.toString();
}
```

Figura 1 – Metoda de trimitere a unui GET

Din câte se observă, metoda dată returnează un string, în care se află însăși răspunsul paginii sub formă de HTTP.

În următoarea metodă, din Figura 2, se află porțiunea de cod, în care se află parsarea paginii propriu zise. Ea se face cu ajutorul librăriei Jsoup. În primul rând se găsește forma de logare după id, și apoi din forma de logare se găsesc toate câmpurile de input. Din câmpurile de input se găsesc acelea care-s responsabile de numele și parola utilizatorului.

Astfel se creează lista de parametri pentru a se pregăti trimiterea unui POST către pagina dorită cu valorile date de programator. Această medotă întoarce lista parametrilor.

```
private List<NameValuePair> getInputs (String page, String username, String password)
        throws UnsupportedEncodingException{
    System.out.println("Looking for data to input!");
    Document document = Jsoup.parse(page);
    Element loginForm = document.getElementById("login_form");
    Elements inputFields = loginForm.getElementsByTag( tagName: "input");

    List<NameValuePair> paramList = new ArrayList<NameValuePair>();
    for (Element inputField : inputFields){
        String key = inputField.attr( attributeKey: "name");
        String value = inputField.attr( attributeKey: "value");

        if (key.equals("username")){
            value = username;
            paramList.add(new BasicNameValuePair(key, value));
        } else if (key.equals("password")){
            value = password;
            paramList.add(new BasicNameValuePair(key, value));
        }
    }
    return paramList;
}
```

Figura 2 – Parsarea paginii

În Figura 3 este formată metoda de trimite a unui POST către serverul web. Iarăși din browserul web se analizează structura antetului necesar pentru POST, și se completează propriu-zis. De notat este faptul că mai este necesar și de trimis parametrii de logare.

```
private void sendPost (String url, List<NameValuePair> postParams) throws Exception{
    HttpPost post = new HttpPost(url);

    post.setHeader( name: "accept",
            value: "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8");
    post.setHeader( name: "accept-language", value: "en-US,en;q=0.8");
    post.setHeader( name: "cache-control", value: "max-age=0");
    post.setHeader( name: "content-type", value: "application/x-www-form-urlencoded");
    post.setHeader( name: "cookie", cookies);
    post.setHeader( name: "origin", value: "https://torrentsmd.com");
    post.setHeader( name: "referer", value: "https://torrentsmd.com/");
    post.setHeader( name: "user-agent", value: "Mozilla/5.0");

    post.setEntity(new UrlEncodedFormEntity(postParams));

    HttpResponse response = client.execute(post);

    System.out.println("\nSending POST request to " + url);
    System.out.println("Parameters: " + postParams);
    System.out.println("Response code " + response.getStatusLine().getStatusCode());

    BufferedReader rd = new BufferedReader(
            new InputStreamReader(response.getEntity().getContent()));

    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = rd.readLine()) != null) {
        result.append(line + "\n");
    }
    System.out.println(result.toString());
}
```

Figura 3 – Trimiterea unui POST

Din răspunsul primit, putem parsa pagina din nou pentru scoaterea unei informații necesare. Acest lucru este prezentat în figura 4. Ideea e aceiași ca la parsarea anterioară.

```
static int counter = 0;
public void getMovies(String page) throws UnsupportedEncodingException{
    System.out.println("Looking for torrent links!");
    counter++;
    Document document = Jsoup.parse(page);
    //System.out.println(document);
    Element table = document.select( cssQuery: "table[class=tableTorrents]").first();
    Iterator<Element> iterator = table.select( cssQuery: "a[href^=/details.php]").iterator();
    Element nextPage = document.select( cssQuery: "a[href^=browse.php?page=" + counter + "]").first();
    while (iterator.hasNext()){
        //System.out.println("Link: " + iterator.next().attr("href"));
        System.out.println("Link: " + iterator.next().text());
    }

}
```

Figura 4 – Parsarea repetată a paginii cu alți parametri

Rezultatul rulării acestor metode întorc răspunsul prezentat în figura 5. Deci în primul rând se trimite un GET către URL-ul ales. Putem observa că conexiunea a avut loc cu succes din cauza răspunsului 200. Apoi are loc parsarea propriu-zisă a paginii, unde se caută input-urile pentru logare. După formarea parametrilor, se trimite un POST către pagină, cu parametrii dați. De notat este că ei vor fi encodați înainte de trimitere. Specific site-ului dat, este că după logare, el face redirect spre pagina principală a utilizatorului, fapt care se observă din răspunsul paginii 302. După ce logarea a avut loc cu succes, se mai trimite un GET către următoarea pagină care va fi parsată, și se scot toate câmpurile necesare.

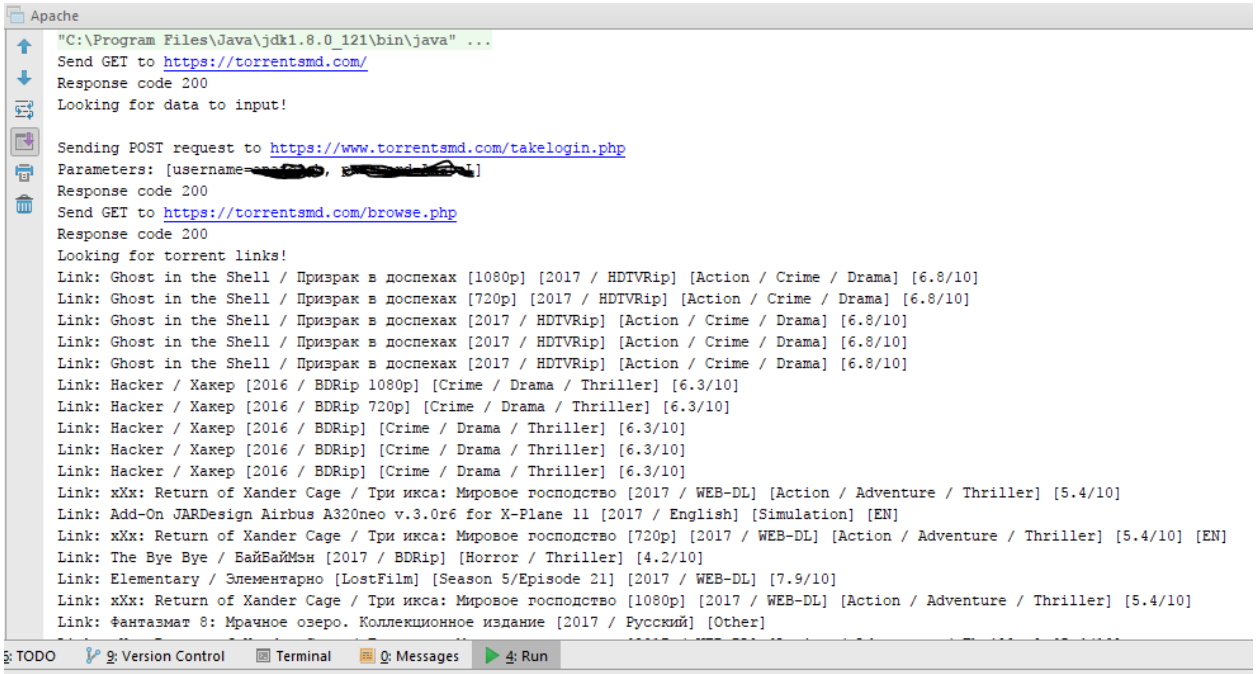

Figura 5 – Rezultatul rulării programului

Aceiași metoda va fi folosită folosind multi-threadingul în figura 6. Pentru aceasta a fost folosit mecanismul din limbajul Java ThreadPool, care va gestiona threadurile.

```
while(true) {
    //System.out.println("Counter is " + counter);
    Runnable worker = new Runnable() {
        @Override
        public void run() {
            //System.out.println("This is thread "+Thread.currentThread().getId());
            process();
        }
        synchronized public void process(){
            try {
                Apache a = new Apache();
                String page = a.sendGet( url: "https://torrentsmd.com/browse.php?page=" + counter++);
                a.getMovies(page);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };
    if(counter == 10){
        executor.shutdown();
        break;
    }
    executor.execute(worker);
}
```

Figura 6 – Implimentarea Multithreadingului

Rezultatul rulării în multithreading este prezentat în figura 7. Astfel putem observa că mai multe threaduri parsează diferite pagini în același timp.

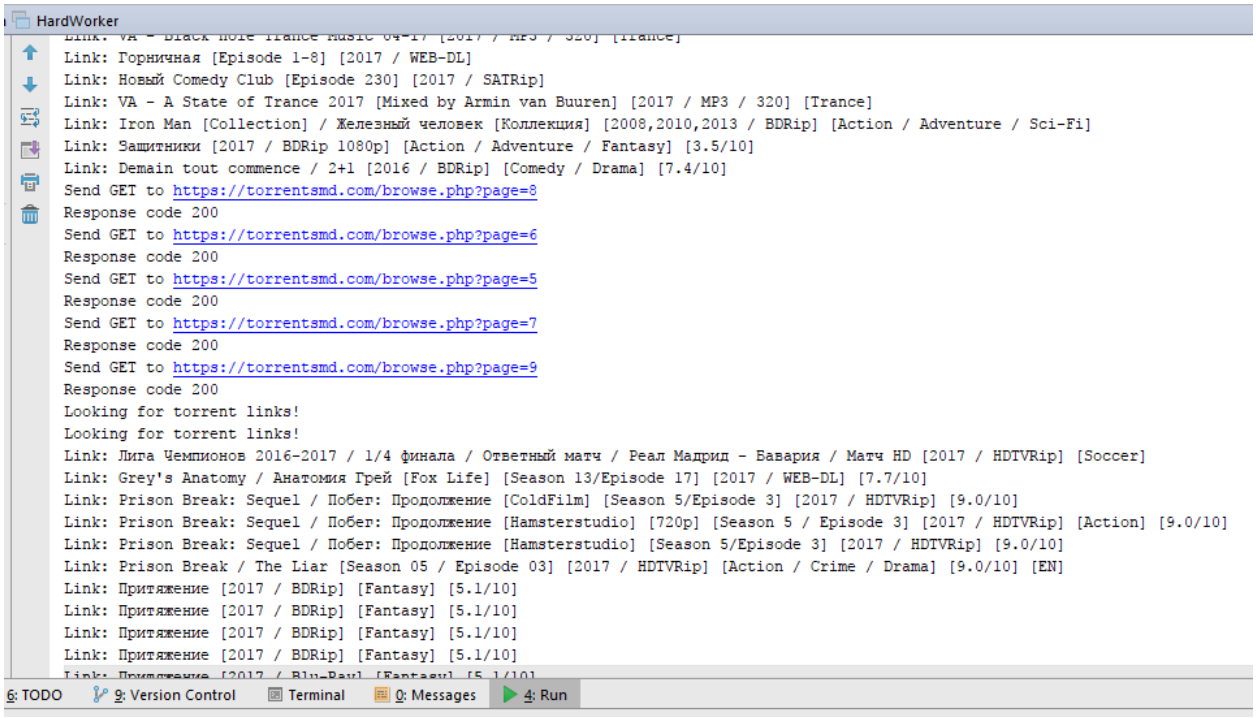

Figura 7 – Rezultatul rulării multithreading

Concluzie

În urma acestei lucrări de laborator au fost obținute abilități de lucru cu metodele HTTP în limbajul Java. A fost înțeles protocolul HTTP și rolul acestuia în comnicarea în WEB. De notat este că au fost și studiate răspunsurile primite de la serviciul Web, cât și tratarea acestor răspunsuri în conformitate cu necesitățile față de ele.

Deasemenea au fost studiate și componentele Java utile în amplicarea protocolului HTTP. A fost utilizat pachetul de componente ApacheHttpComponents. A fost observat că componentele acestei librării pot asigura suportul necesar pentru atingerea obiectivelor lucrării de laborator. A fost observat că ele tratează redirect-urile în mod necesar.

Bibliografie

1. Academia Mikrotik. [Resursă electronică]. Regim de acces: http://www.academia-mikrotik.ro/modelul-isoosi.html

2. Pure Python. [Resursă electronică]. Regim de acces: http://purepython.eaudeweb.ro/wiki/World-Wide-Web-%C8%99i-reprezentarea-datelor

3. Apache HttpComponents. [Resursă electronică]. Regim de acces: https://hc.apache.org/downloads.cgi