Semestrální práce na téma

Zobrazovač HTTP hlaviček

Pavel Jurča <xjurp20@vse.cz>

4IZ110 Informační a komunikační sítě
Praha 2013

Obsah

Cíl práce

Zadání semestrální práce znělo "Zobrazovač HTTP hlaviček." Hlavní funkcí programu tedy má být umožnit posílání modifikovaných zpráv (hlaviček) nad protokolem *Hypertext Transfer Protocol* a následné zobrazení jak právě odeslané hlavičky (request), tak i odpovědi (response) navrácené webovým serverem v návaznosti na náš předchozí požadavek. Parametrizace HTTP requestů byla definována možností upravit verzi protokolu, metodu posílaného požadavku a položek *Referer*, *User-Agent a Accept-Language*. Konkrétní forma implementace byla ponechána na volném výběru zpracovatele, tedy mne. V úvahu připadalo hned několik způsobů zpracovaní.

Program napsat jako desktop aplikaci s grafickým rozhraním (GUI) v některém objektovém jazyku jakými jsou třeba C#.NET nebo Java. Protože by se ovšem jednalo o samostatnou stand-alone aplikaci, kterou je potřeba nejdříve distribuovat (rozeslat uživatelům jednotlivé kopie programu) a rovněž doufat, že specifika jejich operačního systému (OS) neomezí správnou funkcionalitu programu, rozhodl jsem se jít cestou jinou - cestou webové aplikace.

Webová aplikace může být umístěna na libovolný webový server a tím zpřístupněna všem potencionálním uživatelům z celého světa, kteří mají přistup k síti Internet a disponují webovým prohlížečem. Zároveň toto řešení poskytuje snadnou údržbu programu a tím pádem i aktualizace v reálném čase.

Samozřejmě i na poli webových technologií lze zvolit více cest (implementací), které vás více či méně dovedou ke stejnému výsledku. Zaleží tedy na zvážení programátora. Já sám jsem se odebral cestou rozhraní *Common Gateway Interface* (CGI) a prog. jazyka *Practical Extraction and Report Language* (Perl) a frontend řeším pomocí značkovacího jazyka *HTML* s trochou *CSS* (kaskádové styly) a *JavaScriptu*.

Úvod do problematiky

HTTP [1]

Hypertext Transfer Protocol je zavedený jazyk a schéma komunikace mezi webovým prohlížečem na straně jedné a webovým serverem na straně druhé po síti Internet. Je definován konsorciem *World Wide Web Consortium* (W3) a podrobnější informace naleznete na http://www.w3.org/Protocols/.

URI [2]

Jednotlivé zdroje (soubory) jsou identifikovány HTTP schématem *Uniform Resource Identifier* (URI). URI je jen obecná třída, pod kterou spadají lokátory *Uniform Resource Name* (URN) a *Uniform Resource Locator* (URL). V praxi jsou všechny tyto 3 pojmy často zaměňovány, ale můžeme si je představit následující cestou:

Vaše jméno je podobné URN, zatímco vaše adresa bydliště je podobná URL. Oba tyto údaje jsou ale ovšem URI, tedy něco, co vás určitým způsobem identifikuje. Ovšem standardně se toto pojmenovávání moc nerozlišuje a obecně se všem zdrojům na síti Internet (web) referuje jako URL.

URL adresa [2]

URL adresa má své přesné schéma, aby umožnilo navigaci k určitému souboru v tak obrovské síti jakým Internet dnešní doby bezesporu je. URL lokátor může vypadat následovně:

http://www.oreilly.com:80/cgi/calender.cgi?month=july#week3

http je schéma

www.oreilly.com je hostitel, tedy stroj (i virtuální) se spuštěným webovým serverem

80 je port, na kterém služba běží – pokud není definován, použije se defaultní pro dané schéma

cgi/calender.cgi je cesta, která reprezentuje umístění zdroje (souboru, hypertextového doku.)

month=july je dotaz, který se předá odkazovanému zdroji

week3 je fragment (záložka), definující konkrétní řádek v dokumentu

Více informací lze nalézt v RFC dokumentu http://www.ietf.org/rfc/rfc1738.txt.

HTTP hlavička [3]

HTTP hlavička je svým stylem a formátem podobná struktuře elektronického emailu. Historicky je formát HTTP zpráv založen na mnoha konvencích používaných emailem, které byly definovány v *Multipurpose Internet Mail Extension* (MIME) konvenci.

Jednoduchá *HTTP request* hlavička pro webovou stránku *http://www.perl.org/index.html* může vypadat následovně:

GET /index.html HTTP/1.1

Host: www.perl.org

Kde *GET* je metoda požadavku, která serveru na druhé straně sděluje, že chceme zaslat obsah této webové stránky, kterou poté náš prohlížeč zobrazí.

Metod požadavků je samozřejmě více, z těch nejbežnějších zmíním ještě:

HEAD pro server znamená, aby nezasílal obsah stránky

TRACE server žádá, aby tazateli zaslal zpět přesnou kopii HTTP dotazu (*echo back*)

POST znamená, že jsou serveru zasílána nějaká data ke zpracování (třeba z formuláře)

První řádka HTTP hlavičky má unikátní formát a speciální význam. Je zvaná *request line* v požadavku na server a *status line* v odpovědi serveru prohlížeči.

Položky HTTP hlavičky nejsou *case-sensitive*, mohou se objevit v libovolném pořadí a každá řádka musí být vždy zakončena konstantou *CRLF.*

Hlavička rovněž musí být od obsahu HTTP zprávy oddělena novým řádkem (opět *CRLF*).

Pro kompletní výčet všech položek HTTP hlavičky navštivte stránku

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

CGI[4]

Common Gateway Interface je postavené přímo nad protokolem HTTP. CGI je prostředí, které běží ve webovém serveru jakým je třeba http://httpd.apache.org/, a ve kterém mohou být opakovaně spouštěny skripty napsané v jakémkoli programovacím jazyku, který ja na serveru nainstalován. Může to být Visual Basic, Java nebo třeba interpreter BASH dostupný téměř na každém *NIXovém systému. Ale nejčastěji se setkáte s jazykem Perl.

CGI skripty se tedy na webu využívají k vytváření dynamického obsahu, takového, který se třeba mění každou vteřinu – jakým může například být vypisování logu přístupů na webový server.

CGI také umožňuje přístup k nejrůznějším parametrům jak nastavení serveru, tak i údajům o uživateli a jeho prohlížeči, který tento CGI skript zavolal. Toto je uloženo v asociativním poli $ENV.

Například v Perlu lze příkazem 'print $ENV{“HTTP_REFERER”}' vypsat, z jakého URL k nám nový uživatel přišel.

Snad už netřeba zmiňovat, že prohlížeč na straně klienta jednoduše nic neví o tom, jakým

způsobem byl jemu poslaný obsah odpovědi vytvořen – to už je věc serveru a CGI.

Perl

"*Perl is easy to learn because it resembles other popular languages (such as C) [...]*"[5]

"Perl is easily portable and available on many platforms."[5]

"Perl contains extremely powerful string manipulation [...]"[5]

"Perl does not require strict variable types; numbers, strings, and booleans are simply scalars."[5]

"There are countless open source modules for Perl available on CPAN, ranging from modules for creating dynamic graphics to interfacing with Internet servers and database engines."[5]

Zkratka *Perl* v překladu znamená něco jako *praktický jazyk pro dolování a analýzu dat*. Perl vytvořil skvělý programátor Larry Wall.

“*He created Perl in the mid-1980s when he was trying to produce some reports from a Usenet-news-like hierarchy of files for a bug-reporting system, and 'awk' ran out of steam.”*[6]

Jak jsem postupoval

Začal jsem nejdříve pochopitelně *backend*em programu. To znamená, že jsem se v editoru http://www.vim.org pokusil v Perlu napsat datovou kostru programu. Výhodou Perlu je, že je distribuován s opravdu obrovským množstvím všemožných modulů http://www.cpan.org/, které si jen stačí ve skriptu zavolat kouzelným slovíčkem *use*.

Věděl jsem, že potřebuji nějakým způsobem vytvořit *HTTP request*, který opět nějakým způsobem odešlu jako HTTP zprávu po síti, na kterou bych měl následně od webového serveru obdržet odpověď (*HTTP response*).

Pro tento účel jsem si nainstaloval lokální webový server http://httpd.apache.org/, a v souboru ~/apache/congf/httpd.conf jsem povolil spouštění CGI skriptů následujícím způsobem:

```
[...]

ScriptAlias /cgi-bin/ "/home/xjurp20/apache/cgi-bin/"

[...]

<Directory "/home/xjurp20/apache/cgi-bin">
    AllowOverride None
    Options +ExecCGI -Includes
    Order allow,deny
    Allow from all
</Directory>

[…]

AddHandler cgi-script .cgi

[…]
```

Soubor jsem uložil a spustil 2 příkazy:

$ *chmod -R +x /home/xjurp20/apache/cgi-bin/*

$ */home/xjurp20/apache/bin/apachectl graceful*

Nyní už je Apache konečně připraven. Spouštím editor Vim a začínám práci:

$ *vi /home/xjurp20/apache/cgi-bin/request.cgi*

K práci s protokolem HTTP budu potřebovat 3 moduly, konkrétně *LWP::UserAgent*, *HTTP::Request* a *HTTP::Headers*.

```
#!/usr/bin/perl -w
#copyleft 2013 pavel jurca
use strict;
use LWP::UserAgent;
use HTTP::Request;
use HTTP::Headers;
#
my $headers = new HTTP::Headers(
#    Accept             => "text/html, text/plain, image/*",
#    Accept_Language    => "en;q=1.0",
#    Referer            => "10.0.0.138"
);
my $request = new HTTP::Request(
    "HEAD",
    "http://www.perl.org",
    $headers
);
$request->protocol("HTTP/1.1");
#
my $agent = new LWP::UserAgent;
$agent->agent("lwp");
my $response = $agent->request($request);
#
print $request->as_string, "\n";
print $response->protocol, " ", $response->status_line, "\n";
print $response->headers_as_string;
```

Z kódu výše je zřejmé, že vytvoření HTTP požadavku a jeho odeslání je opravdu velmi jednoduché:

V prvním kroku vytvoříme objekt hlavičky, do které můžeme vložit jednotlivé položky HTTP hlavičky stejným způsobem, jako když vytváříme asociativní pole (hash).

Ve druhém kroku vytvoříme samotný požadavek, ve kterém si zvolíme metodu *HTTP requestu,* v našem případě HEAD, poté zadáme absolutní URL adresu, na kterou chceme požadavek odeslat a jako poslední do požadavku vložíme vytvořenou hlavičku z předchozího kroku. Pro specifikaci HTTP protokolu můžeme zavolat metodu '*protocol*' nad objektem *HTTP::Request.*
Například: *$request->protocol("HTTP/1.0");*

Ve třetím kroku už jen vytvoříme *user-agenta,* který náš požadavek odešle. Modul *LWP::UserAgent* tedy simuluje/supluje váš oblíbený webový prohlížeč.

Po spuštení skriptu *request.cgi* v terminálu dostanete stejný výstup, jako je ten na další stránce.
Ačkoliv na první pohled to vypadá, že se vše povedlo, bohužel opak je pravdou. Kód *$request->as_string* by měl vrátit hlavičku našeho požadavku. Ale jak jsme si všimli, chybí nám položka *Host,* která je ve verzi HTTP/1.1 povinná. A jak uvidíme za chvíli, náš *user-agent* nám sice položku *Host* správně přidá i ji skutečně odešle, jen už není žádný způsob, kterým by se šlo té skutečně odeslané hlavičky zpětně dobrat a zobrazit její obsah. My namísto toho vypisujeme náš původní *request,* který ale prostě už není ten, který se vážně odešle serveru.
Druhý problém by nastal, pokud bychom odeslali požadavek HTTP metodou *GET* a chtěli si nechat zobrazit obsah odpovědi. Kód *$response->content* sice při spuštění v terminálu nemá se správným kódováním problémy, ale při zařazení výstupu do HTML stránky už máte z *odpovědi* jen rozsypaný čaj. A to už nemluvím o tom, že tato navrácená data jsou *RAW,* tedy ta, ve kterých nelze nahrazovat nebo vyhledávat, protože nejsou textovým řetězcem.

výstup ze skriptu request.cgi

```
1.  HEAD http://www.perl.org HTTP/1.1
2.  User-Agent: lwp
3.
4.
5.  HTTP/1.1 200 OK
6.  Connection: close
7.  Date: Wed, 08 May 2013 10:10:34 GMT
8.  Via: 1.1 varnish
9.  Age: 30
10. Server: Combust/Plack (Perl)
11. Vary: Accept-Encoding
12. Content-Type: text/html; charset=utf-8
13. Last-Modified: Wed, 08 May 2013 10:10:04 GMT
14. Client-Date: Wed, 08 May 2013 10:10:49 GMT
15. Client-Peer: 207.171.7.41:80
16. Client-Response-Num: 1
17. P3P: CP="There's no P3P policy. Learn why here: http://www.w3.org/P3P/"
18. X-Cache: HIT
19. X-Cache-Hits: 3
20. X-Proxy: proxy1
21. X-Served-By: varnish2.develooper.com
22. X-Varnish: 148232717 148232634
```

skutečně odchycená data na síťovém rozhraní

$ tcpdump -vv -s 1500 -i wlan0 'port 80'

```
4.  ò���'Q JJ8r  sY"   E<b@@@
5.  x3  P  ~
6.  ''Q  JJ"   j8r  sE<=@/  x3
7.  P           A
8.  X     'Q BB8r  sY"   E4b@@@
9.  x3  P   
10. m  X  'Qd  8r  sY"   E b@@
11. x3  P   qU
12.     X HEAD / HTTP/1.1
13. TE: deflate,gzip;q=0.3
14. Connection: TE, close
15. Host: www.perl.org
16. User-Agent: lwp
17.
18. '?BB"   j8r  sE4  @/ Lx3
19. P
20.                
21. Yx    'Q  NN8r  sY"   E@b@@@
22. x3  P     h
23.  `  X
24.         'Q    "   j8r  sE @/  x3
25. P
26.                {
27. Yx   HTTP/1.1 200 OK
28. Server: Combust/Plack (Perl)
29. Vary: Accept-Encoding
30. Content-Type: text/html; charset=utf-8
31. Last-Modified: Wed, 08 May 2013 10:23:06 GMT
32. P3P: CP="There's no P3P policy. Learn why here: http://www.w3.org/P3P/"
33. Date: Wed, 08 May 2013 10:24:34 GMT
34. X-Varnish: 148235292 148235030
35. Age: 88
36. Via: 1.1 varnish
37. Connection: close
38. X-Served-By: varnish2.develooper.com
39. X-Cache: HIT
40. X-Cache-Hits: 7
41. X-Proxy: proxy2
```

Závěr

Jakmile jsem posílání *HTTP requestů* otestoval, čekala mne už poslední fáze této semestrální práce. Nakódovat *frontend* v HTML, aby bylo možné se skriptem pružně reagovat. HTML šablonu jsem nakódoval v editoru https://netbeans.org/. A pomocí formuláře v HTML odesílám metodou *POST* vstupní parametry nového HTTP požadavku skriptu *request.cgi*, který je zpracuje a vrátí hlavičky požadavku (*request*) i odpovědi (*response*) + případný obsah odpovědi.

Poznámky:

[1] Guelich S., Gundavaram S., Birznieks G.: CGI Programming, USA, O'Reilly 2000, s. 16.

[2] tamtéž, s. 17.

[3] tamtéž, s. 24.

[4] tamtéž, s. 42.

[5] tamtéž, s. 8.

[6] Schwartz R., Phoenix T.: Learning Perl, USA, O'Reilly 2001, s. 4.