© Kiskapu Kft. Minden jog fenntartva

Én, a robot

Önműködő webes ügynök Asimov módra.

A Web életünk részévé vált. A böngészés nemcsak gyors és egyszerű módja az információszerzésnek, hanem gyakran az egyedüli is. Ugyanakkor a Weben található ismeretanyag hatalmas mérete és a mindennapi feladatok nyomasztó sokasága miatt az adatszerzést magát jó lenne önműködővé tenni. Ezt az igényt remekül kielégítené Asimov beszélő robotja, amit önálló munkákkal bízhatnánk meg. Amíg azonban nem írunk 2058-at, marad a Perl.
Ez a cikk egy olyan robot elkészítéséről szól, ami meglátogat egy keresőmotort, végrehajtja a keresést, a kapott eredményből kiszűri a hivatkozásokat, és kiírja őket a képernyőre. A bámulatosan egyszerű példán keresztül bemutatom az LWP, URI, illetve HTML-modulok használatát. Ebből kiindulva már összetett űrlapok kitöltését végző ügynökök is könnyedén írhatók.

Ha keresés, akkor Google

A ➲ http://www.google.co.hu/-t fogjuk használni a kereséshez. Ahhoz, hogy megkapjuk a keresés eredményeit tartalmazó oldalt, a robotnak nem kell meglátogatnia a főoldalt. Vessünk csak egy pillantást a forrásra:

```
<form action="/search" name=f>
```

Majd lejjebb:

```
<input maxLength=256 size=55 name=q value="">
```

Nagyjából ennyiből áll az egész űrlap. A fejlécben található JavaScript még beszúr néhány rejtett mezőt, amelyek a karakterkódolást és az ehhez hasonló feladatokat határozzák meg. Ezek most a mi szempontunkból nem érdekesek. Ha a `<form>` elemnek nincs `method` értéke, akkor alapértelmezésben `GET`-et használ, vagyis az űrlap adatai a címben fognak szerepelni. Mivel a szövegbeviteli mező neve `q`, például a kutya szóra való rákeresés így történik:

```
http://www.google.co.hu/search?q=kutya
```

A robotnak tehát első lépésben le kell töltenie a fenti címről a HTML-állományt. Ebben segít az LWP. Debian alatt a libwww-perl csomag része, de természetesen a CPAN-ról is le lehet tölteni.

Az LWP használata

Oldalakon keresztül írhatnék az LWP korlátlan lehetőségeiről, most azonban csak néhány alapfogást mutatok be. Az LWP egy objektumközpontú felület mögé rejti a TCP/IP-s kapcsolatfelvételt, a HTTP-protokollt és a hibakezelést. A programozónak egy oldal eléréséhez alig kell többet tudnia annak címénél. Először létre kell hozni egy úgynevezett felhasználói ügynököt (UserAgent). Ennek az objektumnak a `request` elemfüggvényét kell egy HTTP-kéréshez meghívni. A függvény egy kérésobjektumot vár értékként, amit szintén egy egyszerű `new` utasítással hozunk létre. A visszatérési érték ugyancsak objektum, aminek különböző tulajdonságai tartalmazzák a válasz HTTP-kódját, annak szöveges jelentését stb.

Az URI használata

A liburi-perl része az URI-modul. Segítségével URI-kkal (Uniform Resource Identifier) kapcsolatos műveleteket végezhetünk. Egyszerű objektumközpontú felülettel bír, csakúgy, mint az LWP. Nekünk most azért lesz rá szükségünk, mert a keresési kifejezésünket (a fenti esetben a kutyát) a felhasználó határozza meg. Ezért szükségünk van arra, hogy az ékezetes betűket, szóközöket és egyéb hasonló különleges karaktereket szabványos, címben is használható formára hozzuk. Például egy címben nem szerepelhet szóköz, ezért a %20 kifejezés helyettesíti. E feladatunkat az URI-modullal könnyedén megoldhatjuk. Mindössze létrehozunk egy URI objektumot, majd az `as_string` elemfüggvényt meghívva megkapjuk a kívánt szabványos címet.

A puding próbája

Eddigi ismereteinket összegezve lássunk egy függvényt, ami az értékként megadott szövegre keres rá a ➲ http://www.google.co.hu-n (lásd az *1. listát*).
Fontos megjegyezni, hogy a `$ua->request` függvény meghívásáig semmilyen kapcsolattartás nem zajlik a hálózaton. Elsőként létrehozzuk a kívánt objektumokat, majd egy függvényhívással felvesszük a kapcsolatot a kiszolgálóval, elküldjük a kérést, megkapjuk a választ, és lezárjuk a kapcsolatot. A függ-

```
use LWP::UserAgent;
use URI;

sub search {
    my $page =
       'http://www.google.co.hu/search?q=';
# amit meglÆtogatunk

    my $ua = LWP::UserAgent->new;
# a UserAgent objektum lØtrehozÆsa
    $ua->agent("Mozilla/5.0");
# kicsi f llentØs: azt mondom, Mozilla vagyok

    my $q = URI->new(shift);
# az ØrtØkbıl egy URI objektum

    my $req = HTTP::Request->new( GET =>
       $page . $q->as_string );
# egy kØrØsobjektum

    my $res = $ua->request($req);
# a kØrØs vØgrehajtÆsa

    die $res->message . "\n"
        if $res->is_error;
# ha nem siker lt
    return $res->content; # ha siker lt
}
```

vényt egyszerűen kipróbálhatod, ha a program végére beszúrod a következő sort:

```
print search("kutya");
```

Ekkor a program lefutása után a képernyőn a kutya szóra történő keresés eredményének HTML-oldalát látod.

Szűrjük az eredményt!

Most jön a feladat oroszlánrésze. Meg kell keresni az oldalban az eredmények hivatkozásait. Jelen esetben szabályos kifejezéseket is használhatnánk, mivel a szűrési feltételek viszonylag egyszerűek. Viszont egy összetettebb HTML-oldal esetén a mintaillesztés már nem jelent megoldást, ezért lássuk, mire képes a `HTML::Parser`. Nem nehéz kitalálni, hogy melyik csomag rejtheti a számunkra értékes Perl-modult: a `libhtml-parser-perl`. Az utóbbi csomag több modul gyűjteménye, mi azonban most csak a legfontosabbal foglalkozunk.

A HTML::Parser használata

A modul használatához először létre kell hoznunk egy `HTML::Parser`-objektumot. Mivel a modul nagy múltra tekint vissza, a megfelelőség megtartása érdekében két változatát is elérhetjük a `new` függvény segítségével. A legújabb, azaz a hármas változatot fogjuk használni – ezt azért fontos hangsúlyozni, mert nem működik együtt a korábbi, kettes változattal. Egy HTML-oldal tényleges vizsgálata során az értelmező minden egyes elemnél megáll. Ha olyan elemet talál, amihez eseménykezelő tartozik, meghívja a megadott függvényt, ha nem, az alapértelmezett eljárás szerint jár el. Ilyen elemek az egyszerű szöveg, a nyitóelemek (`<table>`), a záróelemek (`</table>`), az oldalak elején gyakori meghatározások (`<!DOCTYPE ...>`), a megjegyzések (`<!-- ... -->`) stb. Az objektum létrehozása után a `handler` tagfüggvény segítségével adhatunk meg különböző eseménykezelőket. Ezeknek a megadását követően meghívhatjuk a `parse` függvényt, értékként átadva a HTML-oldalt. Ami igazán kellemes a `HTML::Parser` használatában, az az, hogy egy eseménykezelő függvény is létrehozhat, törölhet, illetve módosíthat más eseménykezelőket, így nagy bonyolultságú elemzések is végezhetők.

Szűrés a Google esetében

Vizsgáljuk most meg a Google találati oldalát! A kutya szóra történő keresés eredményének az a részlete, ahol az első találat szerepel:

```
<p class=g><a href=http://www.kutya.net/>
```

A fejléc `<style>` eleméből egyértelműen látható, hogy az oldal CSS-t használ. Ha az egész HTML-dokumentumot alaposan átböngésszük, rájöhetünk, hogy minden találat előtt szerepel egy `<p>` elem, ami a megfelelő formázás elérése érdekében „g” osztályú. Sőt csak a találatok előtt szerepel „g” osztályú `<p>` elem. Ez azért fontos, mert rengeteg hivatkozás van az oldalon, olyanok is, amelyek nem találatok a keresésünkre. Ez a `<p>` elem viszont tökéletes szűrési feltétel. Tehát egy olyan eseménykezelőt kell írni, ami a nyitóelemeket vizsgálja. Ha `<p class=g>`-t talál, akkor kicseréli a nyitóelemek eseménykezelőjét, vagyis saját magát egy újra. Az új eseménykezelő az `<a>` elemeket keresi, és az első találatnál kiírja a `href` értéket, majd visszacseréli magát a régi eseménykezelőre. Ez elsőre talán kicsit nyakatekerten hangzik, de a programkód világos és érthető lesz.

Még egy próba

Ezek után lássuk azt a függvényt, ami egy Google találati oldalból kiszűri az eredmények hivatkozásait. A függvény értéke a HTML-oldal (lásd a 2. listát; 48 CD Magazin/Robot könyvtár). Látható, hogy a `handler` függvény első értékében az eseményhez tartozó függvényt a címével adtam meg. Lehetőség van egyszerűen a nevével hivatkozni rá, én azonban ezt nem tartom helyes programozói módszernek. A `handler` második értéke egy szövegfüzér, ami tartalmazza a eseménykezelő meghívása esetén a megadandó értékeket. Ha például az értelmező a következő elembe fut bele:

```
<p class=g>
```

akkor a `start4href` függvényt három értékkel hívja meg. Az első egy egyszerű „p” szövegfüzér. A második egy asszociatív tömb címe, aminek egyetlen kulcs-érték párja van, nevezetesen a `class-g`. Harmadik, egyben utolsó értéke maga az objektum (a fenti példában a `$p` változó tartalma). Először a `start4all` függvény a nyitóelemek eseménykezelője. Ez egészen addig van így, amíg nem talál egy `<p class=g>`-t, amikor is a nyitóelemek új eseménykezelője a `start4href`. Ez az első `<a>` találatnál kiírja a `href` értékét, majd ismét a `start4all`-t teszi meg eseménykezelőnek.

A kész program

Ahhoz, hogy a fenti két függvényből működő alkalmazást varázsoljunk, csupán egymás után meg kell hívnunk őket:

```
my $content = search(shift);
filter($content);
```

Egyszerűen átadhatod parancssorban a keresési kifejezést, és a program máris tíz találat hivatkozásával gazdagítja a képernyődet. Azok számára, akik nem szeretnek sokat gépelni, a program a CD-mellékleten is megtalálható.

Felhívás egy táncra

Szeretném előre leszögezni, hogy egy ilyen robot nem arra való, hogy százezerszer leadja a szavazatodat kedvenc vagy éppen gyűlölt figurádra valamelyik valóságshowból. Sem arra, hogy a történelemtanárod levélcímét felhasználva hatszáz pornóoldalon jegyeztesd be az illetőt, és eláraszd velük a postafiókját. Ezekkel a támadásokkal szemben meglehetősen kevés mód van a védekezésre. Éppen ezért övön aluli ütésnek számít kihasználni az ehhez hasonló lehetőségeket. Hiszem, hogy ennek a folyóiratnak a közönsége van olyan érett, hogy ha választhat a pusztítás és az építés között, akkor az utóbbi mellé áll. Címemre várom az összes ötletes, új robotot. Az enyémen is fejleszthetsz, ha kedved támad, hiszen egyelőre csak az első tíz találatot írja ki. Sok szerencsét a kísérletezgetéshez!

A cikkhez tartozó listák megtalálhatóak a 48. CD Magazin/Robot könyvtárában.

Fülöp Balázs (xut@freemail.hu)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Az ELTE Radnóti Miklós Gyakorlóiskola tanulója immár ötödik éve. Kedvenc írója Slawomir Mro□ek. Leginkább a számítógépes hálózatok biztonsága érdekli.

© Kiskapu Kft. Minden jog fenntartva