

Tworzenie własnych skróconych URL-i w stylu `tinyurl.com`

Skompresowane linki

Długie URL-e są często przyczyną kłopotów, szczególnie przy umieszczaniu w tekstach artykułów, jak również w poczcie elektronicznej, gdzie długość linii jest często ograniczana przez maksymalny rozmiar linii 78 znaków.

MICHAEL SCHILLI



Kiedy czytałem *Cryptogram*, miesięczny periodyk supergwiazdy bezpieczeństwa, Bruce'a Schneier'a [2], zauważyłem, że typowo długie URL-e do różnych odniesień były zaskakująco krótkie. Wszystkie wskazywały na `http://tinyurl.com` i kończyły się krótkim i zagadkowym ciągiem. Interesujące, jak on to zrobił? Ośrodki, takie jak `tinyurl.com` lub `makeashorterlink.com` oferują darmową usługę, która przechowuje długie URL-e i przypisuje im skróty – kombinacje cyfr i liter. Kiedy przeglądarka przysyła żądanie pobrania strony o danym skrócie, `Tinyurl` przekierowuje ją do docelowego URL. Przywiodło mnie to do zarejestrowania `http://tinyurl.com/28uo8` dla potrzeb linków do niniejszego artykułu (zob. Info na końcu tekstu). Listing 1 (`u - Save as Save Can`) pokazuje jak imitować usługi serwisu `tinyurl.com`.

Cały problem jest całkiem łatwy do zaimplementowania w Perlu. Trwały hash przechowuje unikalne skróty jako klucze i zarejestrowane URL-e jako wartości. Skróty wyrażane są jako kolejne liczby, w odróżnieniu jednak od zapisu dziesiętnego używany jest zestaw małych liter i cyfr, co prowadzi nas do systemu trzydziestozłotkowego (26 znaków i 10 cyfr). Dzięki temu liczby o wielkości miliona i większe mogą być reprezentowane za pomocą zaledwie 4 znaków. Na przykład `4c92` reprezentuje liczbę dziesiętną 1000000 w systemie o podstawie 36. Czterocyfrowe liczby w tym systemie mogą służyć do przechowywania 1679616 różnych wartości – co powinno być dla nas w pełni wystarczające.

Utwardzanie skryptu

Oczywiście skrypt, który chcesz opublikować w Internecie, nie może paść ofiarą pierwszego wrogiego hakera, jaki się tylko pojawi. Wprowadźmy więc kilka rygorów bezpieczeństwa: zezwólmy na rejestrowanie 200 URL-i z jednego adresu IP na dzień – to powinno uchronić nas przed działaniami każdego złooczyńcy próbującego zapełnić nasz twardy dysk bezsensownymi URL-ami. Większy dostawcy usług internetowych, tacy jak np. AOL, używają tego samego adresu IP do obsługi dużej liczby klientów w tym samym czasie. Jednakże w tym przypadku wartość 200 URL-i także powinna być satysfakcjonująca.

- Maksymalny rozmiar pliku bazy danych nie powinien przekroczyć określonej, konfigurowalnej wartości (np. 10 MB). Kiedy ten próg zostałby osiągnięty, skrypt nie powinien przyjmować do zapisu żadnych nowych URL-i, ale nadal powinien umożliwiać wyszukiwanie uprzednio zdefiniowanych URL-i.
- URL nie może przekroczyć długości 256 znaków. Wszystkie dłuższe URL-e powinny być przez skrypt odrzucane.
- Wszystkie zdarzenia powinny być rejestrowane przez skrypt w rotacyjnym pliku dziennika o konfigurowalnym maksymalnym rozmiarze.

Funkcjonalności CGI skryptu realizowane będą za pomocą modułu CGI Lincolna Stein'a i odgałęzienia tegoż modułu `CGI::Carp` z uaktywnionym tagiem `fatalToBrowser`, w celu przechwytywania wszystkich wyjątków i wy-

świetlania ich w przeglądarce dla ułatwienia debugowania. Oczywiście później, po wdrożeniu skryptu do systemu produkcyjnego, należałoby go wykomentować.

Jeśli skrypt nie znajdzie `url` wśród zgromadzonych do tej pory, wyświetli w przeglądarce formularz do wprowadzenia URL-a. Kliknięcie na przycisk `Submit` spowoduje ponowne wysłanie URL-a wprowadzonego przez użytkownika do skryptu jako parametr `url`. Skrypt następnie generuje krótki URL i składa mapowanie pomiędzy skróconą formą i pełnym URL w swojej mini bazie danych, jeśli nie było takiego zapisu dotychczas. Skrócony URL może wyglądać następująco:

```
http://server.com/cgi/u/xxxx
```

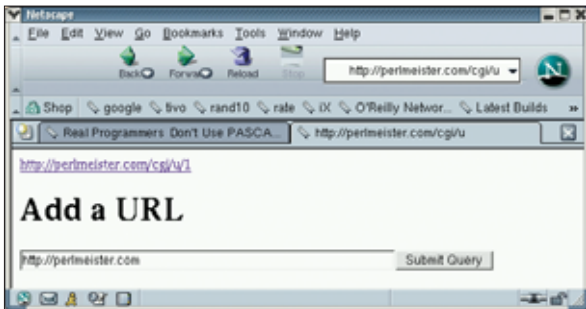
Unikalny ID `xxxx` URL-a doklepany na końcu jest wysyłany do skryptu `u`, gdzie środowisko CGI dostarcza go do zmiennej `$ENV{PATH_INFO}`. Jeśli skrypt otrzyma takie żądanie, wydobędzie odpowiedni, pełny URL z bazy danych i zwróci go za pomocą funkcji `redirect()`, przekierowując w ten sposób przeglądarkę do właściwej strony.

Logi rotacyjne

Obsługa logowania została zrealizowana z wykorzystaniem tandemu modułów `Log::Log4perl qw(:easy)` i `FileRotate` z kolekcji `Log::Dispatch`. Dla zwiększenia wygody zdefiniowano makra `DEBUG()`, `INFO()` i `LOGDIE().size=1000000` ustawia maksymalny rozmiar pliku na 1 MB. `max=1` wskazuje, że `appender` z modułu `FileRotate` ma przełączać cały plik logu o nazwie `shrink.log` do pliku `shrink.log.1`, kiedy `shrink.log` przekroczy granicę progu 1 MB. Dodatkowe kopie nie będą tworzone, dzięki czemu całkowita przestrzeń dyskowa, poświęcona plikom logowym, jest nie większa niż 2 MB.

Trwały hash `%URLS`, dowiązany do pliku `/tmp/shrink.dat` za pomocą funkcji `tie()`, nie przechowuje w `u` pojedynczego odwzorowania klucz-wartość, ale potrójne. Dlatego też dla każdego z kluczy potrzebne są prefiksy:

- `by_shrink/`: skrót do URL-a
- `by_url/`: URL do skrótu
- `next/`: następny skrót do przydzielenia



Rysunek 1. Kompresor oczekuje nowego URL-a, składa URL w bazie danych i generuje skrót.

Istnieje kilka okoliczności, w których skrypt powinien po prostu się zatrzymać – odwiązać trwałe hash i wyjść z programu. Rozwiązanie tego za pomocą *exit* byłoby nieeleganckie, jak również mogłoby powodować problemy w środowisku takim jak *mod_perl*. Z kolei *return()* działa tylko wówczas, gdy *perl* wykonuje podprogram. Ostatecznie zdecydowałem się zastosować stare dobre *goto*, którego Prawdziwi Programiści w odróżnieniu od Zjadaczy Tarty (zob. [3]) nie obawiają się użyć. To dlatego skrypt wykorzystuje *goto*, skacząc do etykiety *END*, która jest zdefiniowana dalej.

Cache pliku ze zdefiniowaną datą wygaśnięcia pomaga ograniczyć liczbę URL-i, które użytkownik może składać dla danego adresu IP w ciągu jednego dnia. Pomysłowy moduł *Cache::Cache* dostarcza prostego interfejsu wykorzystującego funkcje: *set()* do wprowadzania nowych zapisów i *get* do ich wydobywania.

Klasa *Cache::FileCache*, wchodząca w skład modułu, realizuje to za pomocą drzewa plików na dysku.

Krótką pamięć

Dla każdego żądania skrypt inkrementuje licznik związany z danym adresem IP. Kiedy osiągnie on maksymalną skonfigurowaną wartość (200), odmawia obsługi żądań składowania nowych URL-i dla tego adresu. *IP.Cache::FileCache* zapomina o danym adresie IP po dniu nieaktywności, resetując związany z nim licznik do zera. Jednakże przyjęty algorytm nie jest idealny: co gorsza może zablokować dany IP na cały dzień, jeśli liczba zgłoszonych żądań przekroczy ustalony limit.

W celu dostarczenia adresu IP klienta wykorzystywana jest zmienna *\$ENV{REMOTE_ADDR}* środowiska CGI serwera WWW. U uruchomiony z linii poleceń skrypt nie otrzymuje jednakże żadnych wartości dla *\$ENV{REMOTE_ADDR}*. Właśnie dla obsługi takich przypadków występuje proste przedstawienie łańcucha *NO_IP* w linii 148.

Opcja *default_expires_in* konstruktora *Ca-*

che::FileCache określa interwał w sekundach, od ostatniego wywołania funkcji *set()*, po którym cache po prostu zapomina o zapisie. Wartość *true* ustawiona dla *auto_purge_on_get* oznacza, że cache powinien być przeszukiwany pod kątem występowania przeterminowanych zapisów po każdym wywołaniu *get()* i oczyszczany z nich. Chroni to przed nadmiernym opuchnięciem cache'za ze strony akceptowalnych adresów IP.

Wszelchświat systemu 36-kowego

Funkcja *base36()*, zdefiniowana w linii 123 i następnych, konwertuje liczby dziesiętkowe do ich odpowiedników w systemie 36-kowym. Jak to działa? Liczba w systemie dziesiętkowym ma następującą konstrukcję:

$$a*1 + b*10 + c*10*10 + \dots$$

gdzie a, b, c odpowiada cyfrom liczby w odwrotnej kolejności znaczenia. 156 zatem zapisujemy:

$$6*1 + 5*10 + 1*10*10$$

Analogicznie liczby w systemie 36-kowym mają postać:

$$a*1 + b*36 + b*36*36 + \dots$$

Algorytm konwertujący liczbę dziesiętną do liczby systemu 36-kowego jest następujący: wyznacz resztę z dzielenia $d/36$ (to jest d modulo 36 lub $d \% 36$). Wynikiem tego działania jest ostatnia cyfra liczby w zapisie 36-kowym. Następnie weź część całkowitą z poprzedniego działania i znowu podziel przez 36. Otrzymań resztę zapisz do następnej cyfry (od prawej strony do lewej). Poprzednie instrukcje powtarzaj do chwili, gdy część całkowita z dzielenia wyniesie 0.

Funkcja *base36()* definiuje najpierw wszystkie dopuszczalne znaki (tj. cyfry od 0 do 9 i małe litery od a do z) w tablicy *@chars*. Jej rozmiar jest wyznaczany w następujący sposób:

$$\text{my } \$b = @chars;$$

Wynik 36 dla *@chars* w kontekście skalarnym nie stanowi zaskoczenia.

Następnie pętla *for* wylicza modulo *\$num % \$b* dla każdej iteracji, zwracając (od-prawej-do-lewej) kolejne cyfry liczby w zapisie 36-kowym. Instrukcja *\$num /= \$b* w każdym przebiegu pętli *for* wykonuje dzielenie *\$num* przez *\$b* z pominięciem składowej ułamekowej – wymuszone dyrektywą *use integer* w linii 127. Wyrażenie

```
$result .= $chars[$num % $b];
```

wydobywa odpowiedni znak z zestawu znaków systemu 36-kowego i dodaje go do końca łańcucha *\$result* -- liczby lub raczej sekwencji znaków dla systemu docelowego, w odwrotnej kolejności. Korygowane jest to za pomocą wyrażenia:

```
return scalar reverse $result;
```

które odwraca kolejność znaków łańcucha *\$result*. Niezbędne jest wymuszenie kontekstu skalarnego, jako że *reverse* mogłoby odwrócić po prostu kolejność listy skalarów przekazywanych do niego w kontekście listy, pozostawiając każdy z nich bez zmian.

Instalacja

Skrypt wymaga modułów: *Log::Log4perl*, *Log::dispatch::FileRotate* i *Cache::FileCache*, które są dostępne do pobrania z serwisu CPAN. Ścieżki do pliku logu (linia 21) i pliku bazy danych (linia 12) powinny być zaadaptowane, tak by odpowiadały konfiguracji twojego lokalnego środowiska. Sam skrypt powinien być umieszczony w katalogu *cgi-bin* serwera WWW. Jeśli chcesz, możesz uruchomić skrypt z linii poleceń (zwróć uwagę na prawa do wykonania i zapisu dla katalogu danych), ale skrypt powinien generalnie pracować w środowisku webowym – tu jednak należy uważać na ID użytkownika (zazwyczaj jest to *nobody*). Dalej więc, bądź demonem i skracaj te URL-e! ■

INFO

- [1] Listingi do artykułu:
<ftp://www.linux-magazin.de/pub/listings/magazin/2004/02/Perl>
lub <http://perlmeister.com/cgi/lu/2>
- [2] Cryptogram: <http://www.counterpane.com/crypto-gram.html>
lub <http://perlmeister.com/cgi/lu/3>
- [3] Real programmers vs. quiche-eaters,
<http://www-users.cs.york.ac.uk/~susan/joke/quiche.htm>
lub <http://perlmeister.com/cgi/lu/b>

Listing 1. `u` -- Save as Save Can

```

001 #!/usr/bin/perl
002 #####
003 # Mike Schilli, 2003
004 # (m@perlmeister.com)
005 #####
006 use warnings;
007 use strict;
008 use Log::Log4perl qw(:easy);
009 use Cache::FileCache;
010
011 my $DB_FILE =
012     „/tmp/shrinky.dat“;
013 my $DB_MAX_SIZE = 10_000_000;
014 my $MAX_URL_LEN = 256;
015 my $REQS_PER_IP = 200;
016
017 Log::Log4perl->init(\ <EOT>);
018 log4perl.logger = DEBUG, Rot
019 log4perl.appender.Rot=\\
020 Log::Dispatch::FileRotate
021 log4perl.appender.Rot.fileName=
022     /tmp/shrink.log
023 log4perl.appender.Rot.layout=
024     PatternLayout
025 log4perl.appender.Rot.layout.
026     ConversionPattern=%d %m%n
027 log4perl.appender.Rot.mode=
028     append
029 log4perl.appender.Rot.size=
030     1000000
031 log4perl.appender.Rot.max=
032     1
033 EOT
034
035 use CGI qw(:all);
036 use CGI::Carp
037     qw(fatalsToBrowser);
038 use DB_File;
039
040 tie my %URLS, 'DB_File',
041     $DB_FILE, O_RDWR|O_CREAT,
042     0755 or LOGDIE „tie: $!“;
043
044 # First time init
045 $URLS{'next/'} ||= 1;
046
047 my $redir = '';
048
049 if(exists $ENV{PATH_INFO}) {
050     # Redirect requested
051     my $num = substr(
052         $ENV{PATH_INFO}, 1);
053     $redir =
054         $URLS{'by_shrink/$num'} if
055         exists
056         $URLS{'by_shrink/$num'};
057 }
058
059 if($redir) {
060     print redirect($redir);
061     goto END;
062 }
063
064 # Register new URL
065 my $n = base36(
066     $URLS{'next/'}++);
067 INFO "$n: New: $n";
068 $url = url() . "/$n";
069 $URLS{'by_shrink/$n'} =
070     $url;
071 $URLS{'by_url/$url'} =
072     $url;
073
074 print a({href => $url},
075     $url);
076 }
077
078 # Accept user input
079 print h1('Add a URL'),
080     start_form(),
081     textfield(
082         -size => 60,
083         -name => 'url',
084         -default => 'http://'),
085     submit(), end_form();
086
087 END:
088
089 untie %URLS;
090 #####
091 sub base36 {
092     #####
093     my ($num) = @_;
094     use integer;
095     my @chars = ('0'..'9',
096         'a'..'z');
097     my $result = '';
098     for(my $b=@chars; $num;
099         $num/= $b) {
100         $result .=
101             $chars[$num % $b];
102     }
103     return scalar
104         reverse $result;
105 }
106
107 sub rate_limit {
108     #####
109     my ($ip) = @_;
110     $ip = 'NO_IP'
111     unless defined $ip;
112     INFO "Request from IP $ip";
113
114     my $cache =
115         Cache::FileCache->new({
116             default_expires_in =>
117                 3600*24,
118             auto_purge_on_get =>
119                 1,
120         });
121     my $count =
122         $cache->get($ip);
123     if(defined $count and
124         $count >= $REQS_PER_IP) {
125         INFO "Rate-limit: $ip";
126         return 1;
127     }
128     $cache->set($ip, ++$count);
129     return 0;
130 }

```