Zobrazování hlaviček požadavku a odpovědí HTTP serveru

Služba HTTP-service http://http.nektarinka.cz

Autor: Vojtěch Schlesinger xschv07
Předmět: 4IZ110 (skupina 011)
Vyučující: Otakar Pinkas
Datum vytvoření: 12.4.2007
Datum odevzdání: 13.4.2007

Zadání

Mým cílem bylo vytvoření skriptu, který bude schopen zpracovávat HTTP dotazy metodami GET, HEAD, POST, TRACE a OPTIONS a zobrazovat výsledky (včetně hlaviček).

Vycházel jsem z hotových aplikací Web-Sniffer (http://web-sniffer.net/) a Rex Swain's HTTP Viewer (http://www.rexswain.com/httpview.html).

Podívejme se tedy, jaké byly požadavky na funkčnosti mého skriptu:

- program bude pracovat v síti internet, bude i volně dostupný přes prohlížeč
- příjemné uživatelské rozhranní
- volba protokolu HTTP (1.1 nebo 1.0)
- volba typu dotazu (GET, POST, TRACE, HEAD a OPTIONS)
- možnost nastavení vlastní hlavičky user-agent
- volba portu
- zobrazení hlaviček požadavku i odpovědi; zobrazení těla odpovědi

Naopak, s čím jsem už při návrhu nepočítal:

- automatické přesměrování, v případě, že cílová adresa ve skutečnosti směřuje na další
- automatický převod kodování odpovědi
- připojení na zabezpečené servery (funkce využitá v PHP neumí SSL, musela by se využít knihovna CURL a ta neumožňuje jiné typy dotazu než GET a POST; viz dále)

HTTP protokol

HTTP (Hyper Text Transfer Protocol) je internetový protokol určený původně pro výměnu hypertextových dokumentů ve formátu HTML. Používá obvykle port TCP/80, existuje v několika verzích – 0.9 (historická verze), 1.0 a konečně verze 1.1, která je definována v RFC 2616 [2]. Tento protokol je spolu s elektronickou poštou tím nejvíce používaným a zasloužil se o obrovský rozmach internetu v posledních letech.

V současné době je používán i pro přenos dalších informací. Pomocí rozšíření MIME umí přenášet jakýkoli soubor (podobně jako e-mail), používá se společně s formátem XML pro tzv. webové služby (spouštění vzdálených aplikací) a pomocí aplikačních bran zpřístupňuje i další protokoly, jako je např. FTP nebo SMTP.

HTTP používá jako některé další aplikace tzv. jednotný lokátor prostředků (URL, Uniform Resource Locator), který specifikuje jednoznačné umístění nějakého zdroje v Internetu.

K protokolu HTTP existuje také jeho bezpečnější verze HTTPS, která umožňuje přenášená data šifrovat a tím chránit před odposlechem či jiným narušením.

Dotazovací metody

HTTP definuje několik metod, které se mají provést nad uvedeným objektem (dokumentem).
<metoda> <objekt> HTTP/<verze>

GET
Požadavek na uvedený objekt. Je to nejpoužívanější metoda, kterou využívají i prohlížeče, pokud např. zadáte nějakou adresu do adresního řádku.

HEAD
To samé jako metoda GET, ale už nepředává data. Užitečné, když chceme znát jen hlavičky, ale nezajímá nás tělo odpovědi.

POST
Odesílá uživatelská data na server. Používá se například při odesílání formuláře na webu. S předaným objektem se pak zachází podobně jako při metodě GET.

PUT
Nahraje data na server. Objekt je jméno vytvářeného souboru. Používá se velmi zřídka, pro nahrávání dat na server se běžně používá FTP nebo SCP/SSH.

DELETE
Smaže uvedený objekt ze serveru. Jsou na to potřeba jistá oprávnění stejně jako u metody PUT.

TRACE
Odešle kopii obdrženého požadavku zpět odesílateli, takže klient může zjistit, co na požadavku mění nebo přidávají servery, kterými požadavek prochází.

OPTIONS
Dotaz na server, jaké podporuje metody.

Příklad požadavku GET

```
GET / HTTP/1.0
Host: beta.vse.cz
případně další hlavičky
prázdná řádka
```

Pracovní prostředí a volba technologií

Jelikož jsem chtěl, aby moje služba byla volně dostupná komukoliv na internetu a naprostá drtivá většina WWW serverů běží na Linuxu a umím PHP, byl tento jazyk, konkrétně ve verzi 5.1, která obsahuje vylepšený objektový model (vzhledem k PHP4) jasnou volbou.

Vývoj probíhal na mé pracovní stanici s Windows XP, Apache 2.0.55 s PHP 5.1.2. Celý skript jsem nakonec umístil veřejně na svůj server, na kterém momentálně běží distribuce FreeBSD s Apache 2.0.59 a PHP 5.1.6.

Tato aplikace je tedy napsána v objektovém PHP5, pro zobrazení jsem využil XHTML 1.0 strict a CSS pro formátování.

PHP [3]

PHP (Hypertext Preprocessor, původně Personal Home Page) je skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek. Nejčastěji se začleňuje přímo do struktury jazyka (X)HTML.

PHP skripty jsou prováděny na straně serveru, k uživateli je přenášen až výsledek jejich činnosti.

PHP se stalo velmi oblíbeným hlavně díky tomu, že je zdarma, má jednoduchou syntaxi a bohatou podporu knihoven jak pro používání rozličných databázích tak až po zpracování textu, grafiky, nebo také práci se vzdálenými službami.

Návrh uživatelského rozhranní

Při návrhu uživatelského rozhranní jsem vycházel hlavně již z hotových aplikací (které jsem zmínil v úvodu) a také z požadavků. Veškeré proměnné parametry jsem umístil do formuláře, jehož vzhled si můžete prohlédnout na obrázku 1.

obrázek 1

Popis vstupních parametrů

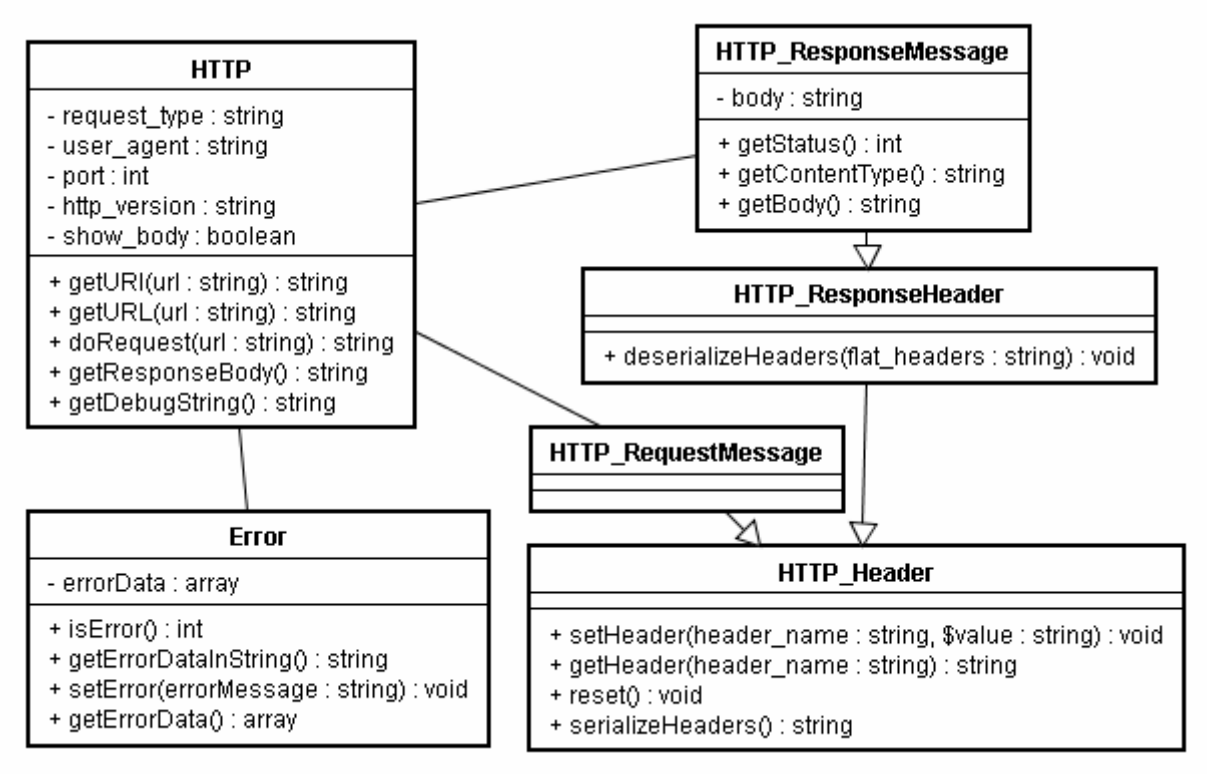
Název parametru	Popis	Povinné	Výchozí hodnota
URL	Adresa, na kterou chceme zaslat dotaz; musí začínat s http://	Ano	http://
Port	Port, na který chceme zaslat dotaz	Ne	80
Protokol	Specifikace verze HTTP protokolu	Ano	HTTP 1.1
Typ požadavku		Ano	GET
User-Agent	Hlavička identifikující prohlížeč, resp. Službu, která se k dané adrese připojí	Ne	User-Agent prohlížece + HTTP-service 1.0
Zobrazit tělo odpovědi	Pokud je zaškrtnuté, zobrazí kromě hlaviček odpovědi i samotné tělo odpovědi	Ne	Zaškrnuto

Po vyplnění povinných parametrů a stisknutí tlačítka provést se uživateli zobrazí samotná odpověď. Vypisuje se jednak IP adresa serveru, na který je směřován dotaz, samotný dotaz a hlavně hlavičky odpovědi. Pokud uživatel zaškrtl možnost „Zobrazit tělo odpovědi“, zobrazí se i samotný obsah vrácené odpovědi.

Návrh programátorského rozhranní

Již jsem psal, že při programování této služby jsem volil objektový přístup. S výhodou jsem využil několik tříd pro reprezentaci dílčích úkonů, úspěšně jsem použil i dědičnost.

Na obrázku 2 je UML Class diagram všech tříd.



obrázek 2

Zde je stručný popis využitých tříd:

Třída HTTP

Hlavní třída celé aplikace. Zajišťuje posílání dotazu a získání odpovědi.

Třída HTTP_Header

Reprezentuje hlavičky protokolu HTTP a nabízí základní metody pro snadnou manipulaci s nimi.

Třída HTTP_RequestMessage

Zaštiťuje celý dotaz. Tato třída nemá žádné metody, proto se může zdát, že je zcela zbytečná, avšak pro budoucí rozšíření aplikace se může hodit a navíc celou aplikaci už nyní zpřehledňuje a zapouzdřuje do dílčích užitečných celků. Je potomkem třídy HTTP_Header.

Třída HTTP_ResponseHeader

Čte hlavičky odpovědi. Je potomkem třídy HTTP_Header.

Třída HTTP_ResponseMessage

Reprezentuje samotnou odpověď. Obsahuje jak tělo, tak i hlavičky odpovědi, je potomkem třídy HTTP_ResponseHeader.

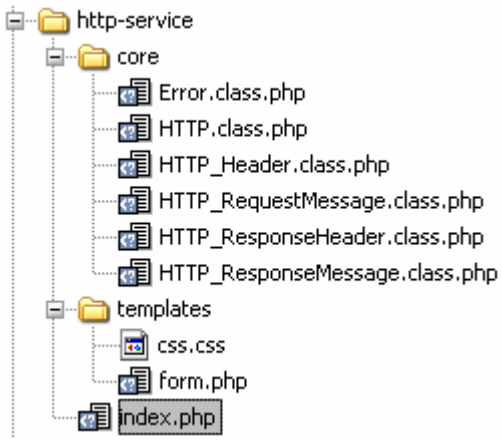
Třída Error

Slouží pro nastavení, uchování a čtení veškerých závažných chyb.

Více o třídách, jejich metodách a atributech v programátorské dokumentaci, která je přílohou elektronické verze této práce.

Fyzická struktura souborů

Fyzickou strukturu souborů aplikace ukazuje obrázek 3.



obrázek 3

Adresář core obsahuje všechny třídy aplikace. Soubory mají přesnou strukturu – NázevTřídy.class.php.

V adresáři templates je uložena velká část prezentační logiky, soubor s kaskádovými styly css.css a také xhtml kod formuláře pro zadání vstupních parametrů – form.php.

Výchozím souborem je pak index.php, který zobrazuje jak samotný formulář, tak i případné odpovědi serveru.

Realizace

Hlavní funkcí v PHP, která se dokáže připojit k vzdálenému serveru na zvoleném portu a zaslat hlavičky je funkce fsockopen() [4].

resource **fsockopen** (string $cílovýServer [, int $port [, int &$čísloChyby [, string &$popisChyby [, float $timeout]]]])

Hlavní nevýhodou této funkce je, že se nedokáže připojit k zabezpečeným serverům. Tento problém by řešila až knihovna CURL [5], ale ta zase nedokáže poslat data jinou metodou než GET a POST a proto jsem od ní v této službě upustil i za cenu nemožnosti připojení na zabezpečené servery.

Příklad připojení a přečtení odpovědi

```
<?php
$fp = fsockopen("www.example.com", 80, $errno, $errstr, 30);
if (!$fp) {
    echo "$errstr ($errno)<br />\n";
} else {
    $out = "GET / HTTP/1.1\r\n";
```

```
    $out .= "Host: www.example.com\r\n";
    $out .= "Connection: Close\r\n\r\n";

    fwrite($fp, $out);
    while (!feof($fp)) {
        echo fgets($fp, 128);
    }
    fclose($fp);
}
?>
```

Programátorská dokumentace

Je součástí elektronické verze této aplikace anebo na adrese http://http.nektarinka.cz/documentation/. Je vygenerována z komentářů v kódu pomocí phpDocumentator. Bohužel tato aplikace má problémy s češtinou, ve které jsem všechny komentáře psal, a tak spíše doporučuji buď vycházet z UML diagramu, anebo shlédnout kompletní zdrojové kódy, které jsou bohatě komentované (např. v příloze).

Otestování aplikace

Chceme zjistit, co nám vrátí tato URL http://seznam.cz/boty na portu 80 metodou GET a ve verzi protokolu HTTP 1.1. Hlavičku User-Agent necháme výchozí.

```
připojování k http://seznam.cz/boty:80 (194.228.32.3) (timeout:20)...
připojeno

Zasílám GET požadavek:
GET /boty HTTP/1.1
Host: seznam.cz
Connection: Close
Pragma: no-cache
Cache-Control: no-cache
Accept: */*
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; cs; rv:1.8.1.3)
Gecko/20070309 Firefox/2.0.0.3 HTTP-service 1.0
Referer: http://http.nektarinka.cz

Čtu hlavičky odpovědi...
Status code: 302
HTTP/1.1 302 Found
Connection: close
Date: Thu, 12 Apr 2007 19:12:04 GMT
Server: Apache/2.0.54 (Debian GNU/Linux)
Location: http://katalog.seznam.cz/boty
Cache-Control: max-age=0
Expires: Thu, 12 Apr 2007 19:12:04 GMT
Content-Length: 213
Content-Type: text/html; charset=iso-8859-1

Čtu tělo odpovědi (4096)...
OK
Odpověď
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
```

```
<p>The document has moved <a
href="http://katalog.seznam.cz/boty">here</a>.</p>
</body></html>
```

Náměty na rozšíření

- možnost zadání formulářových dat (případně i upload souborů) a jejich zaslání metodou POST
- funkčnost metod POST a GET pro zabezpečené servery (kompromis mezi fsockopen() a knihovnou CURL)
- automatické překodování odpovědi
- automatické přesměrování po obdržení příslušných hlaviček (např. 302 Moved Pernamently)

Závěr

Aplikaci jsem testoval zadáním několika serverů a zkoušením i různých metod. Bohužel velká většina mnou známých serverů neumí správně zacházet s jinými metodami než GET, HEAD a POST. Ostatní metody většina běžných WWW serverů nemá ani implementováno. Tím se možnosti mého plného otestování trochu snížily, ale vzhledem k standardizaci zasílaného požadavku nemám větší obavy o nefunkčnost služby.

Celou aplikaci jsem umístil na adresu http://http.nektarinka.cz.

K dispozici je i elektronická verze celé služby na http.nektarinka.cz/http-service.zip.

Použité zdroje

[1] Protokol HTTP http://gama.vse.cz/~pinkas/4iz110/cv7/cv7.html
http://gama.vse.cz/~pinkas/4iz110/cv8/cv8.html
[2] Specifikace HTTP 1.1 http://tools.ietf.org/html/rfc2616
[3] Oficiální stránky jazyku PHP http://php.net
[4] Syntaxe fsockopen() s příklady http://cz2.php.net/manual/en/function.fsockopen.php
[5] Knihovna CURL http://cz2.php.net/manual/en/ref.curl.php

Příloha – zdrojové kódy

HTTP.class.php

```
<?php
/* KONSTANTY */
          define('USER_AGENT',$_SERVER["HTTP_USER_AGENT"].' HTTP-service 1.0'); //user-agent - defaultní hlavička
          define( 'HTTP_CRLF', chr(13) . chr(10)); //oddělovač hlaviček
          define( 'HTTP_V10', '1.0');
          define( 'HTTP_V11', '1.1');
          define( 'HTTP_STATUS_CONTINUE', 100 );
          define( 'HTTP_STATUS_OK',     200 );


/**
 * Hlavní třída HTTP
 * @author Vojtěch Schlesinger <pif@broskev.cz>
 * @package http-service
 * @version 1.0
 *
 */
class HTTP {

          /**
           * Typ požadavku
           *
           * @var string
           */
          public $request_type = 'get';

          /**
           * Hlavička user-agent
           *
           * @var string
           */
          public $user_agent = USER_AGENT;

          /**
           * port dotazu
           *
           * @var int
           */
          public $port = 80;

          /**
           * Verze HTTP protokolu, zadána konstantou, buď HTTP_V10, nebo HTTP_V11
           *
           * @var string
           */
          public $http_version = HTTP_V11;

          /**
           * Zda se má zobrazit i tělo odpovědi
           *
           * @var boolean
           */
          public $show_body = true;

          private $connection; //resource aktuálního připojení
          private $fsock_timeout = 20; //timeout pro připojení
          private $debug_string = ''; //řetězec pro debug zprávy
          private $keep_alive = false; //udržovat připojení?

          private $_request; //instance requestMessage
          private $_response; //instance responseMessage

          public function __construct() {
                    $this->_request = new HTTP_RequestMessage();
                    $this->_response = new HTTP_ResponseMessage();
          }

          /**
           * Zjistí čistě URI (např. /xy) z URL (např. http://seznam.cz/xy)
           * @param string $url adresa začínající na http
           * @return string URI adresy
           */
          public static function getURI($url) {
                    //odstraníme http://
                    $url = str_replace("http://",'',$url);
                    //za prvním lomítkem to je
                    if(strpos($url,"/")==0) return '/';
                    else return substr($url,strpos($url,"/"));
          }

          /**
           * Zjistí čistě hosta (např. seznam.cz) z URL (např. http://seznam.cz/xy)
           * @param string $url adresa začínající na http
           * @return string URL adresy
           */
          public static function getHost($url) {
                    $url = str_replace("http://",'',$url);
                    $bezUri = str_replace(self::getURI($url),'',$url);
                    //a odstraníme http a voilá :)
                    return $bezUri;
          }

          /**
           * Provede samotný dotaz na zadanou adresu podle nastavených parametrů
           * @param string $url adresa na kterou se má poslat požadavek
           * @return string Vrátí tělo odpovědi
           */
          public function doRequest($url) {
                    //musíme zjistit, jaké je URL a jaké je URI
```

```
                $uri = self::getURI($url);
                $host = self::getHost($url);

                //připojíme se k serveru
                $this->_connect($url,$this->port);

                if(!Error::isError()) {
                        $this->_request->setHeader('Host', $host );
                        $this->_request->setHeader('Connection', ($this->_keep_alive?'Keep-Alive':'Close') );
                        $this->_request->setHeader('Pragma', 'no-cache' );
                        $this->_request->setHeader('Cache-Control', 'no-cache' );
                        $this->_request->setHeader('Accept', '*/*' );
                        $this->_request->setHeader('User-Agent', $this->user_agent );
                        $this->_request->setHeader('Referer', 'http://http.nektarinka.cz');

                        $request_type = strtoupper($this->request_type);
                        $this->_debug("\nZasílám $request_type požadavek:\n");

                        if($request_type=='POST') {
                                //ještě nastavíme nějaké hlavičky pro post
                                $this->_request->setHeader('Content-type', 'application/x-www-form-urlencoded');
                                $this->_request->setHeader('Content-length', '0');
                        }

                        $cmd =    "$request_type $uri HTTP/" . $this->http_version . HTTP_CRLF .
                                        $this->_request->serializeHeaders() .
                                        HTTP_CRLF;

                        $this->_debug($cmd);//zaznamáme
                        fwrite($this->connection, $cmd );//zapíšeme
                        return $this->_getResponse($this->show_body);//a zjistíme odpověď
                }
        }

        /**
         * Vrátí tělo odpovědi
         *
         * @return string
         */
        public function getResponseBody() {
                return $this->_response->body;
        }

        /**
         * Vrátí debug string
         *
         * @return string
         */
        public function getDebugString() {
                return nl2br($this->_debug_string);
        }



        /*****************************************************
         * PRIVATE funkce
         */

        /**
         * Připojí se k danému hostiteli na daném portu
         *
         * @param string $url
         * @param int $port
         * @return resource identifikátor připojení
         */
        private function _connect($url, $port) {
                if(Empty($url)) throw new Exception("Není zadána URL");
                else {
                        $this->_debug("Připojování k $url:$port (".gethostbyname(self::getHost($url)).") (timeout:$this-
>fsock_timeout)...");

                        if(!$this->connection = fsockopen(self::getHost($url),$port,$errno,$errstr,$this-
>fsock_timeout)) {
                                Error::setError("Nepodařilo se připojit k $url:$port. (popis chyby: $errno -
$errstr)");
                                $this->_debug(" nepřipojeno!\n");
                        } else {
                                $this->_debug(" připojeno\n");
                        }

                        return $this->connection;
                }
        }

        /**
         * Připojí debug hlášku
         *
         * @param string $string
         */
        private function _debug($string) {
                $this->_debug_string.=$string;
        }


        /**
         * Zjistí odpověď
         *
         * @param boolean $get_body pokud je true, zjistí se i samotné tělo odpovědi, jinak jenom hlavičky
         * @return string tělo odpovědi
         */
        private function _getResponse( $get_body = true ) {
                        //vymažeme hlavičky
```

```
                                $this->_response->reset();
                                $this->_request->reset();
                                $header = '';
                                $body = '';
                                $continue = true;

                                while ($continue) {
                                        $header = '';
                                        $this->_debug("Čtu hlavičky odpovědi...\n");
                                        // přečteme hlavičky
                                        while ( (($line = fgets( $this->connection, 4096 )) != HTTP_CRLF || $header == '') &&
!feof( $this->connection ) ) {
                                            if ($line != HTTP_CRLF) $header .= $line;
                                        }
                                        $this->_response->deserializeHeaders($header);

                                        $this->_debug("Status code: ".$this->_response->getStatus()."\n");
                                        $this->_debug($header);

                                        $continue = ($this->_response->getStatus() == HTTP_STATUS_CONTINUE);
                                        if ($continue) fwrite($this->connection, HTTP_CRLF );
                                }

                                //pokud nechceme odpověď, skončíme
                                if ( !$get_body ) return;
$this->_debug("\nČtu tělo odpovědi (4096)...\n");
                                        while ( !feof( $this->connection ) ) {
                                            $body .= fread( $this->connection, 4096 );
                                        }

                                if(Empty($body)) $this->_debug("\n\nNebyly vráceny žádné data.\n");
                                else $this->_debug("OK");
                                $this->_response->body = $body;
                                return $body;
                        }
}
?>
```

HTTP_Header.class.php

```
<?php
        /**
         * Třída pro práci s http hlavičkami (headers)
         * @author Vojtěch Schlesinger <pif@broskev.cz>
         * @package http-service
         * @version 1.0
         */
        class HTTP_Header {
                private $_headers = array();

                public function __construct() { }

                /**
                 * Vrátí hlavičku podle jména
                 *
                 * @param string $header_name
                 * @return string obsah hlavičky
                 */
                public function getHeader($header_name) {
                        $header_name = $this->_format_header_name($header_name);
                        if(isset($this->_headers[$header_name]))
                                return $this->_headers[$header_name];
                        else
                                return null;
                }

                /**
                 * Nastaví hlavičku
                 *
                 * @param string $header_name
                 * @param string $value
                 */
                public function setHeader($header_name, $value) {
                        if($value != '') {
                                $header_name = $this->_format_header_name($header_name);
                                $this->_headers[$header_name] = $value;
                        }
                }

                /**
                 * Vymaže hlavičky
                 *
                 */
                public function reset() {
                        $this->_headers = array();
                        //$this->_debug     .= "\n hlavičky vymazány \n";
                }

                /**
                 * Upraví hlavičky k použití v http
                 *
                 * @return string
                 */
                public function serializeHeaders() {
                        $str = '';
                        foreach($this->_headers as $name=>$value) {
                                $str .= "$name: $value" . HTTP_CRLF;
                        }
                        return $str;
                }
```

```
                    /**
                     * Provede nezbytné úpravy názvu hlavičky
                     *
                     * @param string $header_name
                     * @return string
                     */
                    private function _format_header_name($header_name) {
                              $formatted = str_replace('-', ' ', strtolower($header_name));
                              $formatted = ucwords($formatted); //první písmeno každého slova je velkým
                              $formatted = str_replace(' ','-',$formatted);
                              return $formatted;
                    }
          }
?>
```

HTTP_RequestMessage.class.php

```
<?php
/**
           * Třída pro zabalení hlaviček pro požadavek
           * @author Vojtěch Schlesinger <pif@broskev.cz>
           * @package http-service
           * @version 1.0
           */
          class HTTP_RequestMessage extends HTTP_Header { }
?>
```

HTTP_ResponseHeader.class.php

```
<?php
          /**
           * Třída pro zabalení hlaviček odpovědi
           * @author Vojtěch Schlesinger <pif@broskev.cz>
           * @package http-service
           * @version 1.0
           */
          class HTTP_ResponseHeader extends HTTP_Header {

                    public function __construct() {
                              parent::__construct();
                    }

                    /**
                     * Rozparsuje hlavičky a uloží je do pole
                     *
                     * @param string $flat_headers hlavičky získané přímo z odpovědi
                     */
                    public function deserializeHeaders( $flat_headers ) {
                              $flat_headers = preg_replace( "/^" . HTTP_CRLF . "/", '', $flat_headers );
                              $tmp_headers = split( HTTP_CRLF, $flat_headers );
                              if (preg_match("'HTTP/(\d\.\d)\s+(\d+).*'i", $tmp_headers[0], $matches )) {
                                        $this->setHeader( 'Protocol-Version', $matches[1] );
                                        $this->setHeader( 'Status', $matches[2] );
                              }
                              array_shift( $tmp_headers );
                              foreach( $tmp_headers as $index=>$value ) {
                                        $pos = strpos( $value, ':' );
                                        if ( $pos ) {
                                                  $key = substr( $value, 0, $pos );
                                                  $value = trim( substr( $value, $pos +1) );
                                                  $this->setHeader( $key, $value );
                                        }
                              }
                    }
          }
?>
```

HTTP_ResponseMessage.class.php

```
<?php
          /**
           * Třída, která zabaluje odpověď
           * @author Vojtěch Schlesinger <pif@broskev.cz>
           * @package http-service
           * @version 1.0
           */
          class HTTP_ResponseMessage extends HTTP_ResponseHeader {

                    /**
                     * Tělo odpovědi
                     *
                     * @var string
                     */
                    public $body = '';


                    public function __construct() {
                              parent::__construct();
                    }

                    /**
                     * Zjistí stavový HTTP kod
                     *
```

```
                 * @return int
                 */
                public function getStatus() {
                        $status = $this->getHeader('Status');
                        if ($status != null )
                                return (integer)$status;
                        else
                                return -1;
                }

                /**
                 * Zjistí typ dokumentu
                 *
                 * @return string
                 */
                public function getContentType() {
                        return $this->getHeader('Content-Type');
                }

                /**
                 * Vrátí tělo odpovědi
                 *
                 * @return string
                 */
                public function getBody() {
                        return $this->body;
                }

                /**
                 * Vymaže tělo a hlavičky
                 *
                 */
                public function reset() {
                        $this->body = '';
                        parent::reset();
                }
        }
?>
```

Error.class.php

```
<?php
/**
 * Třída pro zaznamenání chyb, originálně vytvořena pro framework PWAPI (viz http://pwapi.broskev.cz)
 * @author Vojtech Schlesinger <pif@broskev.cz>
 * @copyright Copyright &copy; 2005-2006, Vojtech Schlesinger
 * @package http-service
 * @version 1.0
 */
class Error {
        /**
         * error things
         *
         * @var array
         */
        public static $errorData = array();

        public function __construct() { }

        /**
         * Do we have some errors?
         * return int, but we can use it like a boolean
         *
         * @return int
         */
        public static function isError() {
                return Count(self::$errorData);
        }

        /**
         * Returns errors in one string
         *
         * @return string
         */
        public static function getErrorDataInString() {
                if(self::isError()) {
                        return '<p>'.implode("<br />",self::getErrorData()).'</p>';
                }
        }
        /**
         * Load new error event
         *
         * @param string $errorMessage error message
         */
        public static function setError($errorMessage) {
                self::$errorData[] = $errorMessage;
        }

        /**
         * Return error data in array
         *
         * @return array
         */
        public static function getErrorData() {
                return self::$errorData;
        }
}
?>
```

css.css

```
body {
          font-family: Verdana, 'Geneva CE', lucida, sans-serif;
          font-size: 0.8em;
          background: #F5F5F5;
          margin: 0 1em 1em 1em;
          border-top: 0.5em solid #E7E7E7;
          color: #333333;
}

h1 {
          margin: 0;
          border: 1px solid black;
          margin: 10px 0 10px 0;
          padding: 10px;
          width: 720px;
}

fieldset {
          width: 720px;
}

.povinne {
          color: red;
          font-size: 80%;
}

#errors {
          color: #EF3B3B;
          border: 1px solid red;
          padding: 10px;
          width: 720px;
}
#debug {
          color: gray;
          border: 1px solid #cacaca;
          padding: 10px;
          width: 720px;
}
```

form.php

```
<?php
define('RADIO_SELECTED'," checked='checked'");
?>

<form method='post' action='index.php'>
<p>údaje označené <span class='povinne'>*</span> je nutné zadat</p>
<fieldset>
          <legend>URL <span class='povinne'>*</span> : port </legend>
          <input style='width: 400px;' type='text' name='url' value='<?php echo !($_POST['url'])?"http://":$_POST['url'];?>'
/> : <input type='text' name='port' size='2' maxlength='4' value='<?php echo !($_POST['port'])?"80":$_POST['port'];?>' />
</fieldset>

<fieldset>
          <legend>Protokol <span class='povinne'>*</span></legend>
          <input type='radio' name='http_version' value='<?php echo HTTP_V11;?>'<?php if(Empty($_POST['http_version']) ||
$_POST['http_version']==HTTP_V11) echo RADIO_SELECTED;?> /> HTTP 1.1<br />
          <input type='radio' name='http_version' value='<?php echo HTTP_V10;?>'<?php if($_POST['http_version']==HTTP_V10)
echo RADIO_SELECTED;?> /> HTTP 1.0
</fieldset>

<fieldset>
          <legend>Typ požadavku <span class='povinne'>*</span></legend>
          <input type='radio' name='request_type' value='get'<?php if($_POST['request_type']=='get' OR
Empty($_POST['request_type'])) echo RADIO_SELECTED;?> /> GET<br />
          <input type='radio' name='request_type' value='post'<?php if($_POST['request_type']=='post') echo RADIO_SELECTED;?>
/> POST<br />
          <input type='radio' name='request_type' value='head'<?php if($_POST['request_type']=='head') echo RADIO_SELECTED;?>
/> HEAD<br />
          <input type='radio' name='request_type' value='trace'<?php if($_POST['request_type']=='trace') echo
RADIO_SELECTED;?> /> TRACE<br />
                    <input type='radio' name='request_type' value='options'<?php if($_POST['request_type']=='options') echo
RADIO_SELECTED;?> /> OPTIONS<br />
</fieldset>

<fieldset>
          <legend>User-Agent</legend>
          <input style='width: 700px;' type='text' name='user_agent' value='<?php echo
!($_POST['user_agent'])?USER_AGENT:$_POST['user_agent'];?>' />
</fieldset>
<br />
<input type='hidden' name='debug' value='1' checked='checked' />
<input type='checkbox' name='showBody' value='1' checked='checked' /> Zobrazit tělo odpovědi
<br />

<input type='submit' name='sended' value=' provést ' />
</form>
```

index.php

```
<?php
/**
 * ***************************
 * HTTP-service
 *
 * skripty slouží k čtení odpovědi přes protokol http různými metodami
 *
 * @author Vojtěch Schlesinger <pif@broskev.cz>
 * @version 1.0
 *
 * */



//nastavíme automatické nahrávání tříd
function __autoload($class_name) {
   require_once './core/'.$class_name.'.class.php';
}

$http = new HTTP();

/**
 * Odeslal data, zkontrolujeme jejich platnost
 */
if(!Empty($_POST['sended'])) {
          //povinné parametry
          if(Empty($_POST['url']) || $_POST['url']=='http://') Error::setError("Nezadali jste URL");
          if(Empty($_POST['request_type'])) Error::setError("Musíte zadat typ požadavku");

          //url musí začínat s http
          if(!ereg("^http://.*$",$_POST['url'])) Error::setError("URL musí začínat s http://, jiné protokoly nejsou
podporovány");

          if(!Error::isError()) {
                    //nejsou chyby, pokračujeme dále
                    $url = $_POST['url'];
                    //fsockopen neumí pracovat s adresou začínající "http://"


                    $http->port = !($_POST['port'])?80:$_POST['port'];
                    $http->request_type = $_POST['request_type'];
                    $http->user_agent = !($_POST['user_agent'])?USER_AGENT:$_POST['user_agent'];
                    $http->http_version = $_POST['http_version'];
                    if(Empty($_POST['showBody'])) $http->show_body = false;
                    $http->doRequest($url);
          }
}


?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs" lang="cs">
  <head>
    <meta http-equiv="Content-Language" content="cs" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>HTTP-service 1.0</title>
          <meta name="Author" content="(c) 2007 Vojtech Schlesinger" />
          <link rel='StyleSheet' type='text/css' href='./templates/css.css' />
  </head>
<body>
<h1>HTTP-service 1.0</h1>
<?php
//jsou nějaké chyby, vypíšeme je
if(Error::isError()) {
          echo "<div id='errors'>".Error::getErrorDataInString().'</div>';
}
?>


<?php require_once 'templates/form.php'; //zobrazíme formulář pro požadavek?>

<?php
//debug výpis
$debugString = $http->getDebugString();
if(!Empty($debugString)) echo "<div id='debug'>$debugString</div>";

$body = $http->getResponseBody();
if(!Empty($body)) {
          echo "<h2>Odpověď</h2>";
          //zobrazíme tělo odpovědi
          echo nl2br(htmlspecialchars($http->getResponseBody()));
}
?>

<p>&copy; 2007 Vojtěch Schlesinger, <a href='http://www.broskev.cz'>webstudio Broskev.cz</a></p>
</body>
</html>
```