Programujeme v PHP / 7. časť: XML funkcie

Na CD REVUE nájdete:
zdrojové kódy

V predchádzajúcich dvoch častiach sme načali tému implementácie XML v PHP. Zistili sme, že aj napriek špecifickým možnostiam aplikácie je XML veľmi silný nástroj. O tom svedčí aj množstvo obslužných funkcií PHP. Druhú časť tohto článku venujeme novej téme – implementácii LDAP v PHP. Vysvetlíme si, čo LDAP znamená a ako funguje.

ČÍTANIE DOKUMENTU XML. Funkcia *xml_parse()* posiela obsah dokumentu XML syntaktickému analyzátoru. Táto funkcia sa postará o analyzovanie dokumentu, pričom zavolá zaregistrované obslužné funkcie. To všetko až po tom, čo nájde uzly v dokumente.

```
Int xml_parse(int parser, string data, int[posledny]);
```

Funkcia vráti hodnotu *true*, ak bola schopná analyzovať vstupné dáta. V opačnom prípade vracia *false*. Chybové hlásenia nám pomôžu zistiť ďalej opísané funkcie *xml_get_error_code()* a *xml_get_error_string()*. Na maličkom príklade si ukážeme použitie funkcie *xml_parse*.

```
<?php
if ( !($fop=fopen($file,"r")) ) {
  die("nie je možné otvoriť súbor $file");
 }
 else {
  while ( $data=fread($fop, 4096) ) {
   if ( !xml_parse($xml_par, $data, feof($fop)) ) {
    die("chyba XML");
   }
  }
 }
?>
```

Po prečítaní treba syntaktický analyzátor uvoľniť. Na to nám slúži funkcia *xml_parser_free()*. Je nevyhnutné uvedomiť si, že takýmto spôsobom uvoľníme len pamäť, ktorú sme alokovali inverznou funkciou k deštruktoru *xml_parser_ create()*.

PARSER A JEHO MOŽNOSTI. Možnosti syntaktického analyzátora môžeme korigovať pomocou dvoch funkcií – *xml_parser_set_option()* a *xml_parser_get_option()*. Prvá voľba *XML_OPTION_CASE_FOLDING* hovorí o tom, že ak hodnota tejto možnosti bude *true*, vtedy budú názvy elementov prevedené pri volaní registrovaných obslužných funkcií na veľké písmená. Druhá voľba *XML_OPTION_TARGET_ENCODING* určuje cieľové kódovanie použité parserom vtedy, keď volá registrované obslužné funkcie.

```
Int xml_parser_set_option(int parser, int moznost, mixed hodnota);
```

Pokiaľ bola nová možnosť nastavená, funkcia vráti hodnotu *true*. Ak volanie zlyhalo, potom *false*. Túto funkciu môžete zavolať kdekoľvek v PHP. Nová možnosť bude platne akceptovaná od miesta jej zadania. Funkcia *xml_parser_ get_option()* vracia hodnotu danej možnosti ako dátový typ návratovej hodnoty, preto je závislá od možnosti. Ak je argument *parser* alebo *moznost* neplatný, je vrátená hodnota *false*.

```
Int xml_parser_get_option(int parser, int moznosti);
```

Konštanty chybových hlásení, ktoré môže funkcia *xml_parse* vrátiť
XML_ERROR_NONE
XML_ERROR_NO_MEMORY
XML_ERROR_SYNTAX
XML_ERROR_NO_ELEMENTS
XML_ERROR_INVALID_TOKEN
XML_ERROR_UNCLOSED_TOKEN
XML_ERROR_PARTIAL_CHAR
XML_ERROR_TAG_MISMATCH
XML_ERROR_DUPLICATE_ATTRIBUTE
XML_ERROR_JUNK_AFTER_DOC_ELEMENT
XML_ERROR_PARAM_ENTITY_REF
XML_ERROR_UNDEFINED_ENTITY
XML_ERROR_RECURSIVE_ENTITY_REF
XML_ERROR_ASYNC_ENTITY
XML_ERROR_BAD_CHAR_REF
XML_ERROR_BINARY_ENTITY_REF
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
XML_ERROR_MISPLACED_XML_PI
XML_ERROR_UNKNOWN_ENCODING
XML_ERROR_INCORRECT_ENCODING
XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING

POMOCNÉ FUNKCIE. Ostatné funkcie na prácu s XML poskytujú užitočné informácie alebo služby, ktoré môžeme potrebovať pri analýze dokumentu XML. Tieto funkcie ponúkajú informácie o všetkých chybách, ku ktorým došlo, ako aj informácie o aktuálnej pozícii v dokumente XML. Funkcia *xml_get_error_code* vracia chybový kód parseru XML. Túto funkciu je vhodné priamo skombinovať s *xml_error_string*, tá už vracia chybové hlásenie zodpovedajúce chybovému kódu.

```
...
$chyba=xml_error_string(xml_get_error_code($parser));
...
```

Funkciou *xml_get_current_line_number()* zistíme číslo aktuálneho riadka z daného syntaktického analyzátora. To znamená, že zistíme číslo riadka, ktorý práve parser spracúva. Podobná funkcia *xml_get_current_column_number()* vracia číslo aktuálneho stĺpca v riadku, ktorý parser spracúva. Posledné dve užitočné funkcie sú určené na kódovanie a dekódovanie pomocou UTF-8 na kódovanie ISO-8859-1. *utf8_decode* a *utf8_encode*.

LDAP. V tejto a nasledujúcich častiach sa bližšie oboznámime s adresárovými službami využívajúcimi protokol LDAP. LDAP (**L**ightweight **D**irectory **A**ccess **P**rotocol) je jednoduchý protokol na prístup k adresárom a ich službám. Protokol LDAP vznikol ako potreba nahradiť existujúcu adresárovú službu poskytovanú protokolom X.500. X.500 bol ťažkopádny možno aj preto, že na potrebnú sieťovú prevádzku využíval sieťový model OSI, ktorý bol na osobných počítačoch nadbytočný. LDAP začínal pod záštitou X.500, no na sieťovú komunikáciu už využíval protokol TCP/IP a umožňoval tak osobným počítačom s implementáciou TCP/IP pristupovať k adresárovým službám. LDAP tiež zrušil množstvo funkcií, ktoré sa využívali v X.500. Aby to nevyzeralo tak, že LDAP vznikol úplne ako nová nezávislá štruktúra, musím pripomenúť, že od X.500 prebral databázový a bezpečnostný model, ktorý postupom času rozšírili.

CHARAKTERISTIKA LDAP. Ako sme si povedali, LDAP je najvhodnejší protokol na prístup k adresárom. LDAP poskytuje adresárové služby takým spôsobom, že správne navrhnutý adresár dovoľuje užívateľom pristupovať k dátam. Povedzme, že jednotky uložené v adresári LDAP sú jednoznačné v tom zmysle, že pri dvoch adresárových jednotkách kdekoľvek na svete nebude identický ich prístupový identifikátor. LDAP je overený štandard, ktorý môže prevziať každý výrobca aj jednotlivec. Práve skutočnosť, že LDAP beží nad TCP/IP, dáva tejto technológii značnú výhodu interkonektivity s počítačmi vo vytvárajúcom sa globálnom internete. Je aj dostatočne pružný, aby ho bolo možné rozšíriť v záujme rôznych aplikácií. Server LDAP používa na ukladanie dát základnú databázu. Nie je však viazaný na konkrétny typ databázy. Na zaistenie zabezpečených transakcií používa overovanie. Na tento účel využíva SASL (Simple Authentication and Security Layer), ktorý umožňuje flexibilitu pri výbere správnej overovacej schémy. Najčastejšie sa na to používa populárny protokol SSL (Secure Socket Layer), vyvinutý spoločnosťou Netscape. LDAP tiež podporuje následníka SSL, a to protokol TLS (Transport Layer Security). SSL a TLS podporuje LDAP od verzie 3 vyššie.

AKO LDAP FUNGUJE? LDAP v zásade napĺňa model klient/server, pričom server LDAP prijíma požiadavky od rôznych klientov a tieto požiadavky obsluhuje. Server LDAP dokáže odoslať odkaz aj na iný server LDAP. Keď hovoríme o LDAP, väčšinou máme na mysli adresárový systém zahŕňajúci nasledujúce tri súčasti. Usporiadanie dát LDAP definuje, ako sú dáta naformátované s ohľadom na komunikujúce jednotky LDAP (klient/server a server/server). Protokol LDAP ako spoločný jazyk, ktorým sa dorozumievajú servery s klientmi, keď klienti pristupujú k adresárom. Protokol LDAP tiež zaisťuje určitú komunikáciu medzi servermi. Klienti LDAP implementovaní s využitím rozhrania API a nástrojov rôznych výrobcov na rôznych platformách, ktorí sa pripájajú k serveru LDAP.

LDAP TERMINOLÓGIA

Položka: je pre adresár to isté ako pre databázu záznam. Uzol obsahujúci meno môže takisto obsahovať ďalšie informácie o nej, napríklad kde býva alebo aká je jej e-mailová adresa atď. Celý uzol nazývame položkou. Zväčša ju nazývame DSE (Distinguished Service Entry).
Atribúty: pre adresár sú tým, čím pre databázu pole záznamu. Atribútom je napríklad opis pri e-mailovej adrese (slovko „mail“).
Objekty: objekty v adresároch môžeme prirovnať k tabuľkám v databázach. Všetky záznamy v databázovej tabuľke sú si podobné v tom zmysle, že majú podobné polia. Objekty majú dodatočnú možnosť rozšírenia a pridania nových atribútov k už existujúcemu zoznamu atribútov.
DIT: Celý informačný strom adresára sa označuje DIT (Directory Information Tree).
Schéma: schéma adresára LDAP poskytuje rozvrhnutie informácií obsiahnutých v adresári a zoskupenie týchto informácií. Takto umožňuje klientom alebo externým rozhraniam určiť, ako sú dáta v adresári usporiadané a ako k nim môžeme pristupovať z hľadiska hľadania, pridávania a odstraňovania... Podrobnosti o triedach objektov a atribútoch LDAP nájdete v dokumente RFC 2256.

Určite ste si všimli, že adresár dávam do protiváhy s databázou. Aby ste neboli prekvapení, prezradím vám, že usporiadanie dát v adresári sa výrazne líši od usporiadania dát v tradičných databázach. Databázy majú obyčajne v záznamoch pole s jedinečnými názvami. To však neplatí v adresári LDAP. Tradičné databázy to dokážu pomocou druhej tabuľky. LDAP však nepotrebuje vytvárať nijakú špeciálnu tabuľku pre viachodnotové atribúty... Adresáre ako LDAP usporiadajú dáta hierarchicky a takisto ich zoskupujú do rôznych skupín. Objekty v adresároch sa významne podobajú tabuľkám. Je to z toho dôvodu, že všetky položky zodpovedajúce objektom rovnakého typu majú podobné atribúty (presne ako záznamy v určitej databázovej tabuľke). Na CD Revue si nájdete zopár vzorových zdrojových kódov, ktoré sa vám pri práci s XML môžu hodiť minimálne ako všeobecný „návod“. Nabudúce budeme naďalej rozoberať LDAP a jeho implementáciu. Teším sa na ďalšie stretnutie s vami.

Jozef KOZÁK ml.

Programujeme v PHP / 8. časť: LDAP

Na CD REVUE nájdete:
zdrojové kódy

Predchádzajúca časť nám pomohla oboznámiť sa s históriou a vývojom LDAP. Napísali sme si čosi aj o jej funkčnosti. LDAP však podporuje aj určité pokročilejšie funkcie, ktoré sa väčšinou nevyužívajú. Vysvetlíme si princíp asynchrónnej operácie. Vezmime si do úvahy aplikáciu, ktorá okrem ďalšieho spracovania dát potrebuje často pristupovať k externým zariadeniam, ako je pevný disk alebo sieť. Operácia s externým zariadením je dosť pomalá a aplikácia musí čakať, dokiaľ sa k danému zariadeniu pristupuje. To až dovtedy, kým zariadenie neodpovie. Medzitým je aplikácia „vyradená" a nemôže vykonávať žiadnu inú činnosť. Toto je model typickej synchrónnej aplikácie! V prípade asynchrónnej operácie volanie funkcie určitého zariadenia funkciu nezablokuje, ale umožní spracúvať ďalšie požiadavky. Aplikácia bude o výsledku dodatočne informovaná vtedy, ak dané zariadenie odpovie. V určitých aplikáciách LDAP sa vyžaduje neustála prevádzka. Na to sa nielen pri LDAP využíva zrkadlenie. Informácie LDAP servera sa „zrkadlia" na ďalší LDAP server. Adresáre LDAP môžu obsahovať citlivé informácie, ako sú súkromné kľúče či heslá. Protokol LDAP preto umožňuje zabezpečenie transakcií pomocou SASL (Simple Authentication and Security Layer). Táto vrstva je dostatočne flexibilná, čoho dôkazom je aj fakt, že využíva rôzne základné metódy šifrovania a overovania. LDAP tiež poskytuje riadený prístup k operáciám, ktoré môžu užívatelia v adresári vykonávať.

ROZHRANIE API LDAP PRE PHP. V tomto článku je pre nás hlavné oboznámiť sa s podporou LDAP v PHP. Podpora je zameraná na zaistenie prístupu k adresárovým serverom LDAP tak, aby sa v aplikáciách vytvorených pomocou PHP mohli využívať ich dáta. Typický klient musí pre prácu s LDAP serverom vykonať zopár krokov. Prvým krokom je použiť funkciu určenú na spojenie so serverom.

```
Int ldap_connect(string[nazov_hostitela], int[port]);
```

Oba argumenty sú voliteľné. Ak nie je zadaný ani jeden argument, bude vrátený identifikátor spojenia nejakého už otvoreného spojenia. Ak zadáte len prvý argument (*nazov_hostitela*), bude použitý predvolený port 389. Pri neúspešnom spojení bude vrátená hodnota *false*.

```
Int ldap_bind(int identifikator_spojenia, string[bind_RDN], string[heslo]);
```

Táto funkcia sa používa na oprávnenie prístupu daného spojenia. Obyčajne sa volá po funkcii *ldap_connect*. Viaže sa k adresáru LDAP so zadaným názvom a heslom. Pri úspechu vracia hodnotu *true*, pri neúspechu *false*. Parametre *bind_RDN* a *heslo* sú voliteľné. Ak argumenty nebudú zadané, bude vykonaný pokus o anonymné prihlásenie. Anonymná väzba je zvyčajne povolená správcom adresárov, ktorý chce umožniť prehľadávanie adresára všetkým, ale bez oprávnenia na úpravy. Ak je povolený anonymný prístup, obyčajne možno získať len obmedzený prístup na čítanie. Medzi tieto operácie zaraďujeme vyhľadávanie, čítanie a porovnávanie atribútov.

```
Int ldap_close(int identifikator_spojenia);
```

Táto funkcia nemá na starosti nič iné ako ukončiť spojenie so serverom LDAP, priradené k zadanému argumentu. Argument *identifikator_spojenia* sa vracia ako výsledok volania *ldap_connect*. Toto volanie je interne identické s volaním *ldap_unbind*. Pri úspechu je v oboch prípadoch vrátená hodnota *true*, v opačnom prípade *false*. Sila LDAP pramení z univerzálnosti operácií vyhľadávania, ktoré možno v adresári vykonať. Existuje niekoľko funkcií, ktoré nielen prehľadávajú, ale aj manipulujú s výsledkami a spracúvajú ich.

```
Int ldap_search(int identifikator_spojenia, string zakladny_DN, string filter, array[atributy]) ;
```

Funkcia vykonáva prehľadávanie adresára so zadaným filtrom s rozsahom pre celý adresár (LDAP_SCOPE_SUBTREE). Vyskytuje sa tu aj voliteľný štvrtý parameter, ktorý možno využiť na obmedzenie atribútov a hodnôt vracaných serverom iba na tie požadované. Používanie štvrtého parametra treba chápať ako dobrú prax. Vyhľadávací filter môže byť jednoduchý alebo zložitý.

```
Int ldap_read(int identifikator_spojenia, string zakladny_DN, string filter, array[atributy]);
```

Funkcia vykonáva prehľadávanie adresára so zadaným filtrom s rozsahom na čítanie položky z adresára (LDAP_SCOPE_BASE). Prázdny filter nie je povolený. Ak chcete prevziať úplne všetky informácie tejto položky, použite filter *objectClass=**. Ak viete, aké typy položiek sa na serveri používajú, môžete použiť príslušný filter.

```
String ldap_dn2unf(string DN);
```

Táto funkcia sa používa len na prevod názvu DN do užívateľsky príjemnejšej formy bez názvu typu.

```
Array ldap_explode_dn(string DN, int [atributy]);
```

Touto funkciou môžeme deliť názov DN vracaný funkciou *ldap_get_dn* na jej súčasti (názvy RDN). Každú časť vo všeobecnosti označujeme ako relatívne rozšírený názov. Práve funkcia *ldap_explode_dn();* vracia pole všetkých týchto komponentov. Atribúty sa používajú na zistenie toho, či sa časti RDN majú vracať iba s hodnotami alebo aj s ich atribútmi. Paradoxom je fakt, že ak chceme dostávať hodnoty s atribútmi, musíme túto premennú nastaviť na *0*.

```
String ldap_first_attribute(int identifikator_spojenia, int id_polozky_vysledku, int ukazovatel);
```

Funkcia *ldap_first_atribute* vracia prvý atribút v položke, ktorá je určená identifikátorom položky. Ďalšie atribúty sa preberajú pomocou funkcie *ldap_next_attribute*. Posledný parameter *ukazovatel* je identifikátor, ktorý poukazuje na miesto v internej pamäti a ďalej sa posúva pomocou odkazov.

```
Int ldap_first_entry(int identifikator_spojenia, int identifikator_vysledku);
```

Položky vo výsledku LDAP sa čítajú postupne pomocou funkcií *ldap_first_entry* a *ldap_next_entry*. Prvá funkcia vracia identifikátor prvej položky vo výsledku. Tento identifikátor sa posunie druhej funkcii, ktorá ďalej odošle ďalšie položky z výsledku. Napokon sa vracia identifikátor prvej položky výsledku (pri úspechu). Pri chybe, samozrejme, parameter *false*.

```
Int ldap_free_result(int identifikator_vysledku);
```

Touto funkciou sa uvoľňuje pamäť interne vyhradená na uloženie výsledkov predchádzajúcej operácie hľadania, na ktorú odkazuje identifikátor výsledku. Všetka rezervovaná pamäť sa uvoľní po skončení skriptu. V prípade, ak skript opakovane zadáva hľadanie rozsiahlej množiny výsledkov, je lepšie volať funkciu *ldap_free_result* a obmedzovať tak využívanie pamäte skriptom pri behu. Pri úspechu je vrátená hodnota *true* a pri neúspechu *false*.

```
Array ldap_get_attributes(int identifikator_spojenia, int id_polozky_vysledku);
```

Funkcia *ldap_get_attributes* sa používa na zjednodušenie čítania atribútov a hodnôt z nejakej položky. Vrátená hodnota je viacrozmerné pole atribútov a hodnôt. Len čo je vyhľadaná určitá položka v adresári, môžeme pomocou tohto volania zistiť, aké informácie obsahuje. Najefektívnejšia aplikácia tohto volania je v takej aplikácii, ktorá prechádza položky v adresári, aj keď nepozná ich štruktúru. Pri úspechu sa vracia úplná informácia položky vo viacrozmernom poli alebo pri neúspechu hodnota *false*.

```
String ldap_get_dn(int identifikator_spojenia, int id_polozky_vysledku);
```

Pomocou tejto funkcie môžeme vyhľadávať názov DN akejkoľvek položky vo výsledku. Pri chybe vracia false.

```
Array ldap_get_entries(int identifikator_spojenia, int identifikator_vysledku);
```

Na zjednodušené čítanie viacerých položiek vo výsledku a následné čítanie atribútov a viacerých hodnôt používame funkciu *ldap_get_entries*. Celá informácia sa volá jediným volaním vo viacrozmernom poli. Zoznam atribútov sa konvertuje na malé písmená. Vracajú sa úplné informácie položky vo viacrozmernom poli a pri neúspechu hodnota *false*.

```
Array ldap_get_values(int identifikator_spojenia, int id_polozky_vysledku, string atribut);
```

Túto funkciu používame na čítanie všetkých hodnôt atribútov v danej položke z výsledku. Položka je určená parametrom *id_polozky_vysledku*. Počet hodnôt možno zistiť hodnotou *count* výsledného poľa. LDAP umožňuje zadanie viacerých položiek jednému atribútu, takže môže napríklad obsahovať množstvo adries v jedinej adresárovej položke určitej osoby (to všetko označené atribútom *mail*).

```
Int ldap_list(identifikator_spojenia, string zakladny_DN, string filter, array [atributy]);
```

Pri prehľadávaní LDAP musíme určiť základ stromu, kde sa má hľadanie začať. Určiť rozsah hľadania znamená vymedziť časť stromu, ktorá sa pri hľadaní použije. Funkciou *ldap_list* sa vykoná prehľadávanie so zadaným filtrom v adresári s rozsahom *LDAP_SCOPE_ONLEVEL* (hľadanie vráti informácie, ktoré sú na úrovni bezprostredne pod základným názvom DN zadaným vo volaní).

```
Int ldap_count_entries(int identifikator_spojenia, int identifikator_vysledku);
```

Takto zistíme počet položiek určených ako výsledok predchádzajúcej operácie hľadania. Parameter *identifikator_vysledku* určuje interný výsledok LDAP. Pri chybe vracia *false*.

```
String ldap_next_attribute(int identifikator_spojenia, int id_polozky_vysledku, int ukazovatel);
```

Funkciu voláme v záujme získania atribútov nejakej položky. Interný stav tohto ukazovateľa udržuje parameter *ukazovatel*. Funkcii sa posúva odkazom. Prvé volanie sa uskutoční s parametrom *id_polozky_vysledku* po volaní *ldap_first_attribute*. Pri úspechu je vrátený atribút položky, pri neúspechu *false*.

```
Int ldap_next_entry(int identifikator_spojenia, int id_položky_vysledku);
```

Funkcia vracia identifikátor nasledujúcej položky vo výsledku, pričom položky sa začínajú čítať pomocou funkcie *ldap_first_entry*. Aj keď adresáre LDAP by svoj obsah nemali meniť často, existujú aj funkcie zadávajúce zmeny v adresári. Okrem úprav PHP ponúka funkcie pridávania a odstraňovania dát na serveri LDAP. Teraz sa v skratke na takéto funkcie pozrieme.

```
Int ldap_add(int identifikator_spojenia, string DN, array polozka);
```

Touto funkciou môžeme pridať do adresára nové položky. Parameter *identifikator_spojenia* je parameter použitý ako návratová hodnota funkcie *ldap_connect*. Pridávaná

položka potrebuje nový názov DN, ktorý zadávame ako druhý. Tretím argumentom je pole skladajúce sa z atribútov a hodnôt novej položky.

```
Int ldap_mod_add(int identifikator_spojenia, string DN, array polozka);
```

Funkcia *ldap_mod_add* pridáva hodnoty atribútov k existujúcim atribútom zadaného názvu DN. Úprava sa vykonáva na úrovni atribútov (nie na úrovni objektov). Pridávanie na úrovni objektov sa uskutočňuje pomocou funkcie *ldap_add*. Pri úspechu vracia hodnotu *true*, pri neúspechu *false*.

```
Int ldap_mod_del(int identifikator_spojenia, string DN, array polozka);
```

Touto funkciou môžeme odstrániť hodnoty atribútov z existujúceho zadaného DN. Úprava je vykonávaná na úrovni atribútov. Odstraňovanie na úrovni objektov sa vykonáva funkciou *ldap_del*.

```
Int ldap_mod_replace(int identifikator_spojenia, string DN, array polozka);
```

Funkciou *ldap_mod_replace* môžeme nahradiť atribúty zadaného DN. Úpravu vykonáva na úrovni atribútov. Pri úspechu vracia hodnotu *true*, pri neúspechu *false*.

```
Int ldap_delete(int identifikator_spojenia, string DN);
```

Odstraňuje určitú položku v adresári LDAP, určenú názvom DN. Táto funkcia je obyčajne vyhradená pre malý počet užívateľov podľa zadania v zozname riadenia prístupu ACL (Access Control List) servera LDAP.

```
Int ldap_modify(int identifikator_spojenia, string DN, array polozka);
```

Úpravy dát LDAP sú obvykle vyhradené pre overených užívateľov. Zoznam ACL umožňuje rôznym užívateľom upravovať rôzne atribúty (všetci užívatelia môžu meniť svoje heslá). Zvláštnu pozornosť treaé venovať nahradzovaniu viachodnotových atribútov. Pretože ak nahradíte nejaký atribút s viacerými hodnotami jedinou hodnotou, nahradíte tým všetky jeho hodnoty. Štruktúra funkcie je rovnaká ako pri *ldap_add*. V tejto časti sme sa oboznámili s použitím adresárových služieb LDAP. Ich použitie si určite obľúbi nie jeden programátor. Na CD Revue nájdete zopár zaujímavých príkladov praktických ukážok, ako LDAP funguje v praxi. Toto miesto sme využili skôr na prezentáciu možností API LDAP v PHP. V budúcej časti sa začneme venovať generovanej grafike v PHP. Teším sa na ďalšie stretnutie s vami.

Jozef KOZÁK ml.

PHP je vo finále / 5. časť: Abstraktné triedy a rozhrania

Na CD REVUE nájdete:
príklady

Piata verzia PHP sa dostala do finále v polovici júla 2004. Na stránkach PC REVUE sme sa začali venovať možnostiam, ktoré ponúka nový skriptovací stroj, už v etape, keď bol k dispozícii druhý kandidát na uvoľnenie. Zamerali sme sa na podporu pre objektovo orientované programovanie. Podrobne sme preskúmali ochranu prístupu k členským premenným a funkciám triedy kľúčovými slovami public, protected a private, overili sme novozavedené špeciálne funkcie triedy (__construct, __destruct, __call, __set, __get), predstavili sme možnosti použitia statických a konštantných členov triedy. V tomto článku budeme pokračovať v skúmaní nových čŕt objektovo orientovaného programovania v PHP5. Sústredíme sa na abstraktné triedy a rozhrania. Dopracujeme sa k nim cez poznanie možností opakovaného využitia kódu dedičnosťou a kompozíciou.

DEDIČNOSŤ A KOMPOZÍCIA. Je všeobecne známe, že objektovo orientované programovanie má zabezpečiť vyššiu produktivitu práce programátorov. To sa má dosiahnuť tak, že vytvorený kód možno využiť opakovane. Samozrejme, opakované využitie raz vytvoreného kódu sa dá dosiahnuť aj v podobe funkcií. Vytváranie a využitie tried a ich inštancií – objektov – však ponúka oveľa dômyselnejšie možnosti na opakované využitie vytvoreného kódu. Dve základné možnosti, ktoré máme pri tom k dispozícii, sú dedičnosť a kompozícia.

O dedičnosti hovoríme vtedy, ak novú triedu vytvárame tak, že tá rozširuje možnosti už vytvorenej triedy. Ak v novej triede využívame objekty existujúcej triedy, hovoríme o kompozícii.

Rozhodnutie o tom, či použiť dedičnosť alebo kompozíciu, si v mnohých prípadoch vyžaduje istú dávku skúsenosti a nie zriedka aj intuície. Je to preto, lebo oba prístupy môžu viesť k rovnakému výsledku. Ukážme to na jednoduchých príkladoch.

Majme triedu CA – je vo výpise 1. Má konštruktor a členskú funkciu FA. Podobne ako v predošlých článkoch aj tu v konštruktore a členskej funkcii triedy je iba výpis textu. Slúži na to, aby sme počas experimentov mohli na základe získaného textu posúdiť, ktoré časti kódu sa dostali k slovu.

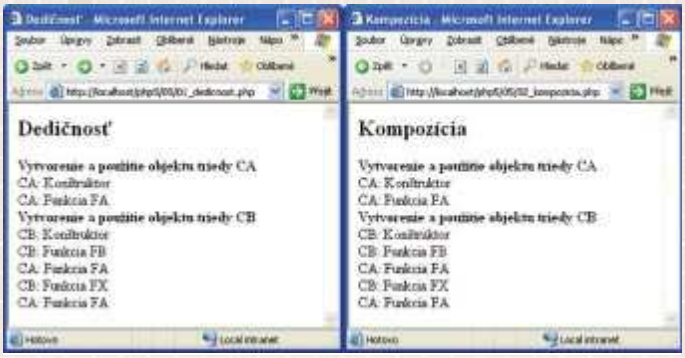
Výpis 1: Zdrojový kód triedy CA

```
1   class CA
2   {
3     public function __construct()
4     {
5       echo "CA: Konštruktor<br />\n";
6     }
7
8     public function FA ()
9     {
10      echo "CA: Funkcia FA<br />\n";
11    }
12  }
```

Vo výpise 2 je zdrojový kód triedy CB, v ktorom je využitá dedičnosť. Očakávame, že trieda CB zdedí funkciu FA a navyše implementuje funkciu FB a FX. Funkcia FB je nezávislá od rodičovskej triedy. Vo funkcii FX je volaná funkcia FA rodičovskej triedy (riadok 16).

Vo výpise 3 je zdrojový kód triedy CB, v ktorom je využitá kompozícia. V riadku 3 je deklarácia členskej premennej $A. Jej hodnota je určená v konštruktore - prebratá je z rovnomenného argumentu. Aj v tomto prípade má trieda CB funkciu FB aj FX. Vo funkcii FX je využitá členská premenná $A na volanie funkcie FA triedy CA.

V tab. 1 je uvedený spôsob využitia triedy CA aj oboch verzií triedy CB. Je tam aj zodpovedajúci text, ktorý je spojený s vytvorením objektu a volaním ich funkcií. Obrázok 1 zachytáva výsledky získané skriptami, ktorých úplné zdrojové texty nájdete na priloženom CD.



Obr. 1 Využitím dedičnosti a kompozície možno dosiahnuť rovnaký výsledok

Zmeny v kóde, ktoré sú vyvolané použitím dedičnosti alebo kompozície, vidieť v treťom a piatom riadku tab. 1. V treťom riadku pri kompozícii konštruktoru triedy CB odovzdávame objekt triedy CA, zatiaľ čo pri dedičnosti konštruktor triedy CB nemá argumenty. V piatom riadku je kód, ktorým je volaná metóda FA triedy CA. Pri dedičnosti môžeme priamo volať túto metódu, lebo trieda CB ju zdedila. Pri kompozícii ju vyvoláme prostredníctvom členskej premennej $A.

NÁSTRAHY KOMPOZÍCIE. V uvedenom príklade použitia kompozície je všetko v poriadku. Je to preto, lebo vieme, že konštruktor triedy CB preberá objekt triedy CA, a rešpektujeme to. Veľká sila a jednoduchosť PHP je daná tým, že nejde o prísne typový jazyk. V jednej a tej istej premennej môže byť číslo, text aj objekty rôznych tried. S našou triedou CB, využívajúcou kompozíciu, získame bezchybne fungujúci skript, ktorý bude obsahovať tieto príkazy:

```
$B = new CB(77);
$B->FB();
```

Konštruktoru CB nie je odovzdaný objekt triedy CA, ale číslo 77. Skriptovací stroj neidentifikuje problém a spokojne vykoná aj príkaz v druhom riadku. Problémy nastanú, keby sme chceli využiť metódy objektu v členskej premennej $A, t. j. keby sme použili volanie:

```
$B->A->FA();
```

Vtedy bude skriptovací stroj protestovať a dostaneme oznam:
Fatal error: Call to a member function FA() on a non-object

Tab. 1 Porovnanie použitia triedy využívajúcej dedičnosť a kompozíciu

n	Dedičnosť	Kompozícia	Vypísaný text
1	$A = new CA();	$A = new CA();	CA: Konštruktor
2	$A->FA();	$A->FA();	CA: Funkcia FA
3	**$B = new CB();**	**$B = new CB($A);**	CB: Konštruktor
4	$B->FB();	$B->FB();	CB: Funkcia FB
5	**$B->FA();**	**$B->A->FA();**	CA: Funkcia FA
6	$B->FX();	$B->FX();	CB: Funkcia FX
			CA: Funkcia FA